A PROGRAM GENERATOR

FOR DESIGNERS

A thesis

submitted in fulfilment

of the requirements for the degree

of

Master of Engineering

(Mechanical)

in the

University of Canterbury

by

A. A. Hunt

University of Canterbury

1988

CONTENTS

ABSTRACT

A computer-aided design package has been developed which will enable an engineering designer without conventional computer programming skills to generate and run a program to be used as a tool in solving a design problem. In contrast to traditional programs the design system is based on decision tables, which allow improved documentation, communication and modification of design logic. Decision tables have previously been used as a programming medium either as input to a specialised processor for conversion into conventional program code, or retained as data to be scanned by a general purpose processor for checking design constraints. The current package advances the use of decision tables as data inputs by automating the preparation of decision table-based application programs. The resulting general purpose processor is menu-driven and application independent. All program information, including equations, is stored and used by the system as data. The Program Generator incorporates both system and design parameter data manipulation features to maximise the flexibility a designer is given to control the program logic as well as parameter values used to reach a solution to a design problem. A worked example on beam bending and listings of all programs are included.

CHAPTER I

INTRODUCTION

The application of computers as design aids has been wide-ranging in both scope and magnitude. Computer-aided design (CAD) continues to grow, fueled by ongoing developments of both hardware and software packages. While a large number of packages have been written for engineering users, the range of engineering tasks is so great that specialist packages can only provide computer-aid for a comparatively small proportion of frequently encountered uses. To harness the resources of the computer for tasks for which specialised software is not available, the engineer must either have a package custom developed to his specifications or, in the case of tasks too small or infrequently performed to warrant the expense or lead time of a commissioned program, he must write his own software. To write his own programs the engineer must be proficient in computer programming and, furthermore, must be able to justify the time that writing programs requires. Regardless of source, specialised programs tend to be rigidly structured and reflect the style and preferences of a particular designer, thus limiting the ability of another user to choose his own approach to solving a problem. Any changes to program packages needed because of changes in user requirements or source material (such as codes, standards or specifications) will require further input from a programmer.

As computer technology has become an integral part of the engineering workplace, so it is gaining significance in the training of engineers. Lack of programming skills also restricts use of computers as

design tools by students in the university learning environment. A university teacher setting a problem which requires the use of a computer must either require all students to be proficient in computer programming, or else provide a program into which students can feed their data. Providing a pre-written program will test only the ability of a student to enter data at a terminal, and will not demonstrate his knowledge of the methods required to solve the problem. The tendency to use computer programs as 'black boxes' and automatically accept their output as correct increases the likelihood of failing to detect errors from incorrect data entry or inappropriate assumptions. As stressed by Smith (1986) in a comment considering whether engineering CAD users are "missing the 'big picture'", it is important for engineers to be familiar with methods used in solving a problem. Development of this familiarity and good engineering judgement is important in engineering practice and must be emphasised in the education of engineers. The ideal, then, for the learning environment, is a system which will enable the student to generate a computer program which reflects his knowledge of a solution method rather than of programming techniques. The reduction of the need for programming skills would make such a system suitable for much wider application in the professional environment.

It is difficult to remove the tendency to use a program from any source as a black box, so it is important to develop clear documentation of the program method. The inflexibility of traditional program structures can be avoided by use of decision tables to represent the design logic. Goel and Fenves (1971) have demonstrated the suitability of decision tables to document specifications and have used decision tables in a general purpose processor to check design constraints. As a

progression from such use of decision tables it would be valuable if the preparation of decision table-expressed application programs could be automated through the use of a general purpose program generator. This would require further development of techniques used to process (run) the generated application programs.

The aim of this project was to investigate the feasibility of developing an application-independent interactive computer package that would enable a designer without conventional computer programming skills to generate and run a program based on decision tables. Such a package would be of particular interest in the university environment.

To avoid the respective roles of the user and writer of the Program Generator being confused, for the purposes of this thesis the writer will be refered to as the systems programmer, and the user of the system as the user or designer. Systems programs are those which form the Program Generator itself, and application programs are those which are generated by the system.

CHAPTER II

THE REQUIREMENTS OF A PROGRAM GENERATOR

Increases in computer availability and usage have not always been matched by system-user interfaces which reflect the widening range of people having direct access to a computer. The primary requirement of the Program Generator is for it to provide a CAD tool to engineers who do not have great computer programming expertise. This requires an interactive system capable of informing the user of all its input requirements, whether they be data inputs or instructions. Using pull-down menus of available options is an effective method of giving the user control of the system without needing to key in specific instructions.

There is generally not just a single solution to a design problem, nor is there often only a single course to a solution. While the design process can be highly formalised the designer will normally exhibit a considerable amount of individualism or art, even if it is exercised within constraints imposed by management policy, standards or codes of practice. The Program Generator must therefore provide a flexibility of approach to allow the designer to develop and test alternative solutions within any restrictions which apply. When a design is commenced, not all the requirements of an acceptable solution may be apparent, as some of the restrictions will depend on the form of the design itself. Rather than following a single model right from the start, the designer may choose to compare a number of low investment, coarse models before selecting one, or possibly more, for further

development. Thus, the system should facilitate the formation and development of computer models with the minimum investment of time or specialised skills. The initial analysis may result in a change of course in solving the problem in hand and, probably, a refinement of the model in use. Knowledge of the problem may also increase on a broader scale, resulting in changes to codes and standards. Hence, the Program Generator must provide straightfoward updating of logic and design parameters to accomodate progressive development of a particular model as well as influences on the design environment. Ready access to, and control over, the program structure and data input is also required to test the sensitivity of a solution to changes in logic or the values of input parameters.

Ideally, the model expressed in the form of a computer program should be able to be easily communicated to other engineers, whether for checking purposes or for subsequent use or modification.

In summary then, the Program Generator For Designers should be a CAD tool sufficiently flexible to allow the designer to exercise his art while also allowing easy updating and development of the expressed logic. Fulfilment of these objectives should allow an increase in design productivity by virtue of quicker setting up and updating, and/or an improvement in design quality because of the greater number of design iterations possible for the same costs incurred by traditional methods. The increased cost-effectiveness of a Program Generator in comparison with previous CAD techniques may even allow the use of CAD to extend to small scale or one-off problems in which computer use would not have previously been financially justifiable.

CHAPTER III

EXPRESSING DESIGN LOGIC

Design of an engineering system is an iterative process comprising two basic aspects: creativity by which the system is proportioned so that the constraints imposed on the system are satisfied; and analysis whereby system performance is predicted on the basis of a mathematical model. Even though the creative process can be highly individual, computers can still assist the designer in quantifying his intentions, particularly where the design belongs to a family of similar forms or where there are strong interactions between groups of parameters.

The Program Generator is built on the conceptual framework provided by Fenves (1972). The terms 'ingredience' and 'dependence', used here to express the relationships within the design logic network, are from the same source.

1. ELEMENTS OF THE DESIGN LOGIC NETWORK

The design logic network is made up of inter-related data elements. The basic relationship between the data elements may be described as a transformation (Fenves, 1972) Ti, of the form

$$T_i : y_i = f_i I_i(y_i)$$

where: $I_i = (x_{i1}, x_{i2}, \ldots x_{in})$,

y_i is an output variable,

x_{ij} are input variables,

f is a functional relationship between input variables.

The transformations T_n may take the form of functions or logical decisions, and may be based on proven theory or subjective opinions. Individual x_{in} may be called ingredients of y_i and, hence, I_i, the collection of ingredients, may be called the ingredience list of y_i. Conversely, as the value of y_i depends on each x_{ij} on its ingredience list I_i, y_i may be called a dependent of each x_{in}. An input variable may be a ingredient of more than one output variable and have more than one dependent. The collection of all the dependents of an input variable x_{ik} may be called the dependence list of x_{ik},

$$D(x_i) = (y_{i1}, y_{i2},...y_{im})$$

For example, a direct stress, S_d,might be calculated from the applied force, F, and the cross sectional area, A. F and A are thus ingredients of S_d, and S_d is a dependent of both F and A.

The value of y_i may be determined by one of a number of functions; the relevant function is identified by the logic of the problem and each function has its own set of ingredients. For example, the support and loading conditions of a beam determine which function used to determine the bending moment.

Because of the direct relationship between ingredience and dependence, only one of the two need be supplied as the other can be internally generated. Ingredience lists are implied in the equations used in calculation, and hence are the simpler of the two to supply.

When a design problem is computerised, flow charts are commonly used to describe the program logic. However, the flow chart becomes unwieldy and confusing with large or complex programs and is not readily updated. Like flow charts, the derived programs are often inflexible and have poor readability, with reliance placed on the programmer to provide sufficient documentation. These factors may make the system production and maintenance costs higher than the costs incurred in actually running the program.

Decision tables which express the design problem's logical conditions, possible actions, and the logic linking the two in a tabular form are an alternative to flow-chart based programs and documentation. Decision tables are more readable and concise than flow charts, hence relationships between variables are more readily apparent and may be easily updated and maintained.

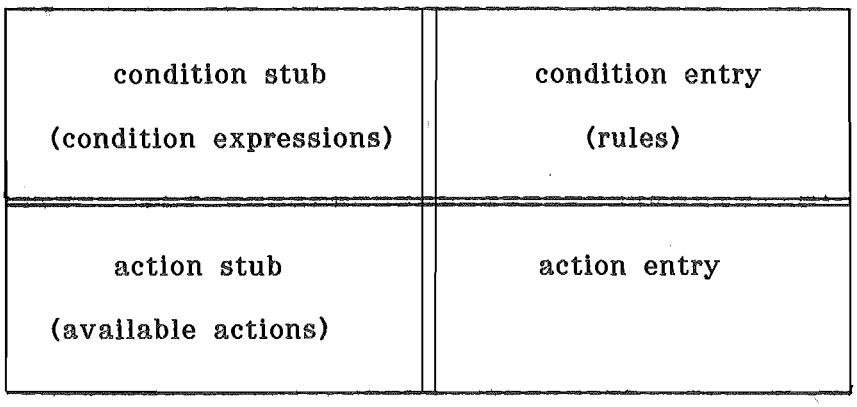


Figure 3.1 Decision table construction

A decision table consists of four parts as shown in figure 3.1 and a sample decision table is shown in figure 3.2.

The condition stub lists the logical conditions on which a decision is based. The action stub lists all the actions which may be taken as a result of the decision. An action might, for example, be the assignment of a value to a variable (either directly or through the application of a particular equation) or the execution of another decision table. The condition entry lists all relevant combinations of

left end free?	Y	-	-	-	-	-
left end fixed?	-	-	-	Y	-	-
left end guided?	-	Y	-	-	-	Y
left end simply supported?	-	-	Y	-	Y	-
right end fixed?	Y	Y	Y	Y	-	-
right end simply supported?	-	-	-	-	Y	Y
M=Wl	X					X
M=Wl/2		X				
M=0.1924Wl			X			
M=Wl/8				X		
M=Wl/4					X	

where: M = bending moment

W = force exerted by point load

l = beam length

Figure 3.2 A sample decision table (maximum beam bending moment)

the values of the logical conditions in columns. Each column of the condition entry is called a 'rule'. The elements of the condition entry may be Y (yes), N (no), or - (immaterial). (Some texts use Y, N, and I, respectively.) The action entry indicates the actions to be taken should each particular rule apply. Action entries may be X, indicating that the action in the same row of the action stub as the mark is to be carried out, or blank if the corresponding action is not to be carried out. (Some texts use Y and blank, respectively.) Decision tables using only Y, N, and - (or their equivalents) in the condition entry are called limited entry tables. McDaniel (1968) gives further information on extended entry tables.

In any given situation only one decision table rule may apply but any number of actions may be executed as an outcome of that rule.

In situations in which more than one rule fits the state of the condition stub the table is said to be ambiguous (Fenves, 1966). As table rules are scanned from left to right for an appropriate rule the left-most rule that fits applies; any subsequent rules are not examined.

A decision table is said to be complete if rules are stated for all combinations of the logical variables in the condition stub.

A rule that applies irrespective of the state of the condition stub is known as an 'else' rule. Else rules are often used to detect error conditions caused by incorrect or incomplete logic, or input data exceeding system capabilities. Else rules may be specified explicitly in a decision table by making all entries in the right-most rule equal to -

(immaterial) or may be implied by a processing algorithm with the ability to detect a 'no rule applies' state.

Dependence relationships can arise through logical as well as functional transformations as shown in figure 3.3.

With repect to figure 3.3, data element A is dependent on conditions C2 and C4 if rule i applies, and on condition C1 if rule j applies. Element A is not, however, a dependent of C3 as both entries in row 3 are immaterial.

	i	j
C1	-	Y
C2	Y	-
C3	-	-
C4	N	-
A =	X	X

Figure 3.3 Dependences from logical relationships.

The advantages of decision tables to describe design logic can now be better appreciated. Firstly, the logic which applies to a particular problem is clearly displayed in a compact form with all the pertinent conditions, actions and logic grouped in adjacent areas. Thus,

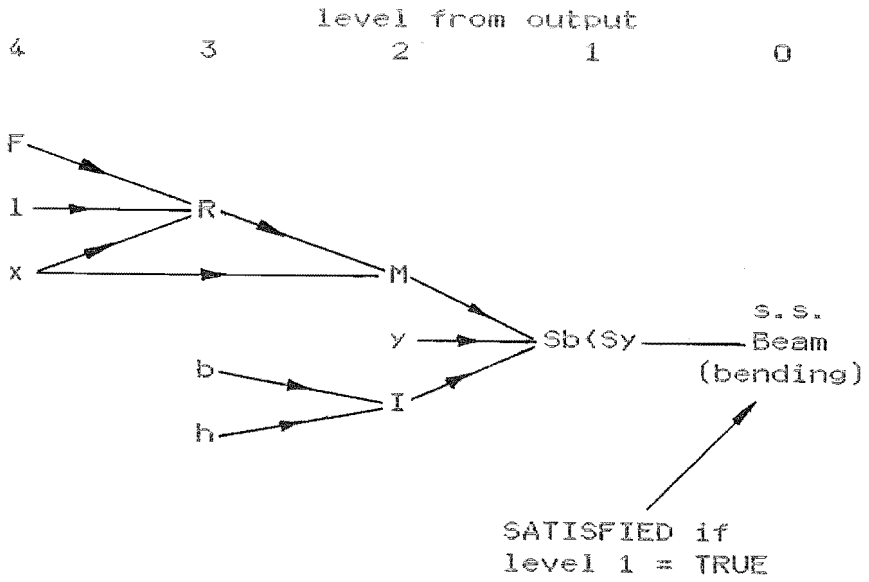
the completeness of the table can be readily assessed and any ambiguities or other errors identified. Secondly, unlike the flow chart, additions and alterations do not alter the clarity and compact nature of decision tables. This provides a distinct advantage in applications such as computer program documentation where accurate communication of the design logic is essential. Thirdly, decision tables can be easily read, analysed and written by non-computer programmers more readily than can flow diagrams. Fourthly, the standard layout of decision tables offers the potential for formalised preparation and processing, including direct use as a programming language. These characteristics, particularly the latter two, are the major reasons for using decision tables as the basis for the Program Generator For Designers.

2. THE GLOBAL DATA NETWORK

As an output variable from one transformation may in turn serve as an input variable to another transformation, supplying the ingredience list for each element provides sufficient information to completely describe the data network.

Two wider relationships can now be identified: the global ingredience and global dependence of a data item. The global ingredience of a data item consists of all the data items influencing it, its ingredients, the ingredients of its ingredients, and so on. Global ingredience is obtained by traversing the network from right to left until all the data elements which may be accessed have been reached. The global dependence of a data item consists of all data elements depending on it, and is obtained by traversing the network from left to right

until all the data elements which may be accessed have been reached.



where:

Sb	bending stress
Sy	yield stress
M	bending moment
y	distance N.A. to farthest fibre
I	second moment of area
R	reaction force
F	applied force
l	beam length
x	distance along beam from support
b	breadth of beam
h	height of beam

Figure 3.4 Data network for a simply supported rectangular beam

The logical expressions which make up decision table condition stubs are themselves elements of the global data network and, so, link the logical transformations described by the decision tables to the data required for the execution of that logic.

With respect to figure 3.4, each element on the network can be described as being a certain level from output. As a data element may be an ingredient to a number of dependents the level relative to each particular output may be different, hence within the global network Fenves (1972) defines the global level from output as being that which gives the longest path from the particular data item to any output item.

3. DESIGNER-CONTROLLED VARIABLES

Within the network there exist input items which have values controlled directly by the designer; these will be refered to as 'designer-controlled variables'. The majority of input variables are designer-controlled; the remainder have values assigned as the outcome of a decision table.

CHAPTER IV

PROCESSING OPTIONS

1. DECISION TABLES

Once a design problem has been expressed in decision table form there are two options for use of decision tables as programming aids. The first is to convert the decision tables into a conventional tree-structured program, either manually or by using a programmed preprocessor. More than one tree-structured program can be derived from any particular decision table depending on the sequence in which the programmer chooses to arrange the design condition expressions from the condition stub. The second option is to retain the decision table in its original form, store it as data, and use it as input to a general purpose interpretive program. This second approach offers greater flexibility than the first, as updating or altering the problem logic does not affect any of the interpretive routines. The tree-structured approach offers a potential basis for a program generation strategy, assuming that a computerised preprocessor is used, but it lacks flexibility and requires considerably more processing programs of greater complexity than does retaining the decision tables as data. The 'store-and-use-as-data' option has the added advantage of neatly fitting in with strategies for storing and using the elements of the wider data network.

An advantage of using decison tables to express design problem logic and then retaining this form for use by a general purpose

processor is that the clarity of logic provided by decision tables renders the additional computer program documentation required by conventional programming methods unnecessary.

2. DATA NETWORK

To execute a decision table one proceeds from a given set of input data, evaluates the logical variables of the condition stub, and then carries out the actions indicated by the appropriate rule. As the set of input data will not contain all the information required to complete execution, some strategy is required for the evaluation of this other data. To overcome this problem 'direct' or 'conditional' execution may be used.

Direct execution reflects traditional flow chart program layout which requires data to be arranged and processed in a predetermined order. Execution begins with the input data and moves 'bottom-up' to derive data at higher levels from input. Execution halts if any data item encountered is not defined at the time it is required. In terms of the network illustrated in figure 3.4, the data network must be evaluated from left to right.

Conditional execution leaves the computer program to decide the sequence the process follows according to what data is or is not defined at the time. This 'top-down' logic allows execution to begin even if some ingredient data is not defined. Thus, if while evaluating some data item it is found that an ingredient of that item is also not defined, processing of the the first item is temporarily suspended, while the

second is evaluated, and is then resumed.

The conditional execution approach has several advantages over the direct approach:

1. Immaterial condition entries in a decision table mean that not all conditions may need to be evaluated for the decision table to be executed. Under some circumstances it may not even be possible to supply a complete set of input data.

2. Optional data, which may remove the need to evaluate some lower level data, is more readily accomodated.

3. Any updating or correction of design constraints or other data requires little effort compared with the possible reprogramming and compilation required if a direct approach is adopted. This is of particular importance within an interactive program generation strategy where maximising the 'user-friendliness' of the system requires that such alterations be easily achieved.

4. The most natural sequence is 'top to bottom' in line with the logic of the problem itself as reflected in the ingredience lists. Hence, once a problem is expressed within the framework outlined in Chapter III, conditional execution is more easily accomplished.

The above points are relevant to computer-aided design per se as well as to program generation. Data input for the Program Generator can be more easily and logically achieved using a top-down strategy, so conditional execution was adopted.

CHAPTER V

DATA STRUCTURES

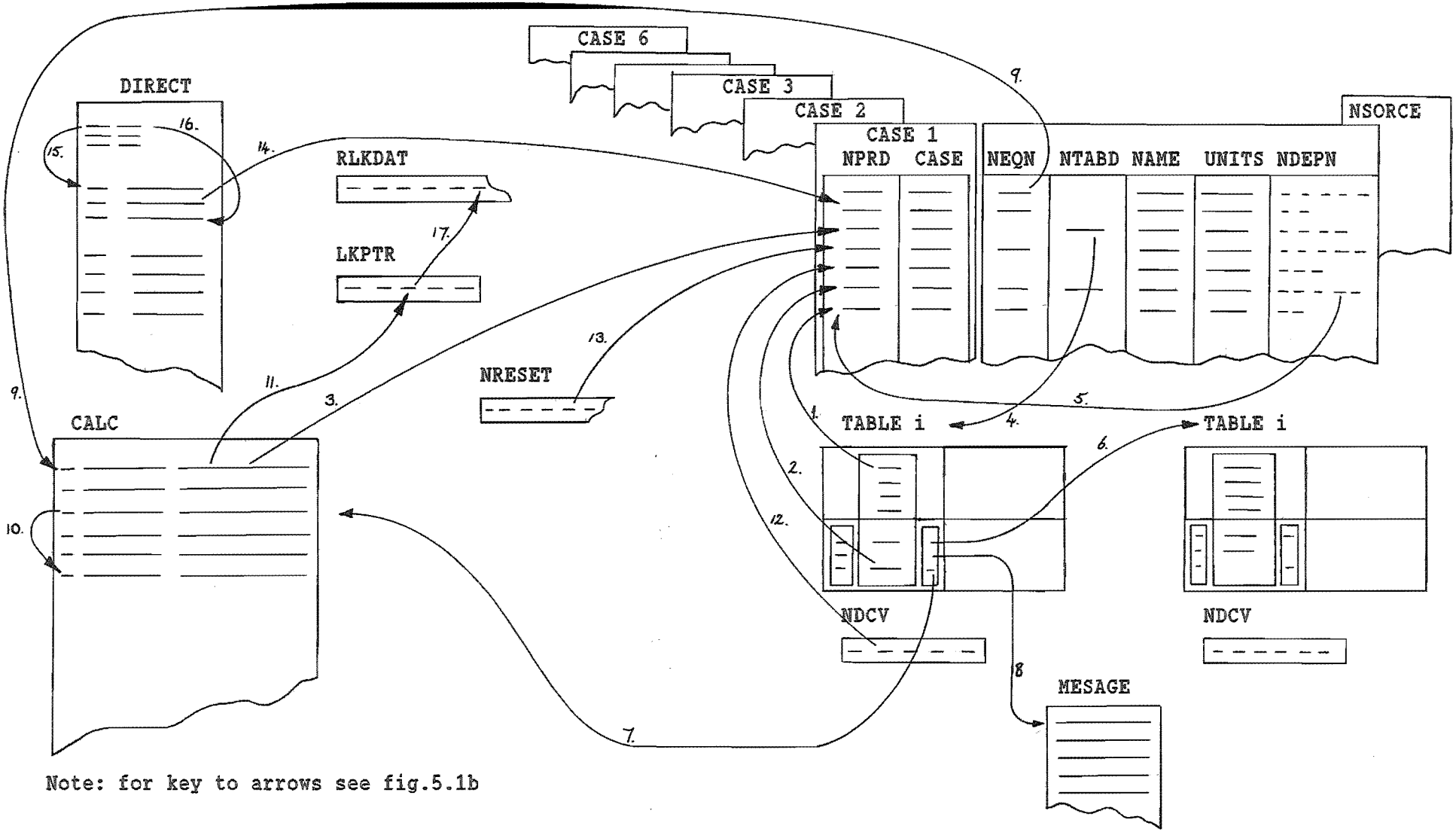
The relationship between decision tables and stored data is shown in Figure 5.1. The names of storage areas shown are those used by the program system programs for the equivalent arrays. A leading 'N' is often used in a name to indicate an integer array or variable to the system programs which use FORTRAN implicit naming conventions.

1. DECISION TABLE STORAGE

Figure 5.2 is complementary to figure 5.1 and represents the detail of the stored decision table.

Condition and action stubs are not stored in their original form by the system. The condition stub is used to store pointers (NSTORE) which indicate the storage locations of the state of each of the conditions (arrow 1, fig. 5.1).

The action stub is used to record the type of action (NTYPE), the number of the equation to be used if the action type specifies calculation of a variable (METHOD), and pointers indicating the storage



Note: for key to arrows see fig.5.1b

Figure 5.1a Decision table - data interaction

ARROW

1. NSTORE (condition stub) to storage location in CASE.
2. NSTORE (action stub) to storage location in CASE.
3. Equation entry pointing to storage location in CASE.
4. NTABD pointing to table to be conditionally executed.
5. Entry in dependents list to storage location in CASE.
6. METHOD pointing to table to be directly executed.
7. Eqn. to be used in direct calculation of a variable.
8. METHOD pointing to mesage to be given.
9. Eqn. to be used in conditional calc. of a variable.
10. Linked list to next equation for same variable.
11. Equation entry pointing to look-up table number.
12. Designer-controlled var. list entry to storage locn.
13. Reset list entry pointing to NPRD to be reset.
14. Directory entry pointing to storage location.
15. Directory header indicating beginning of block.
16. Directory header indicating end of name-length block.
17. Pointer to first storage location for look-up table.

Figure 5.1b Key to arrows in figure 5.1a

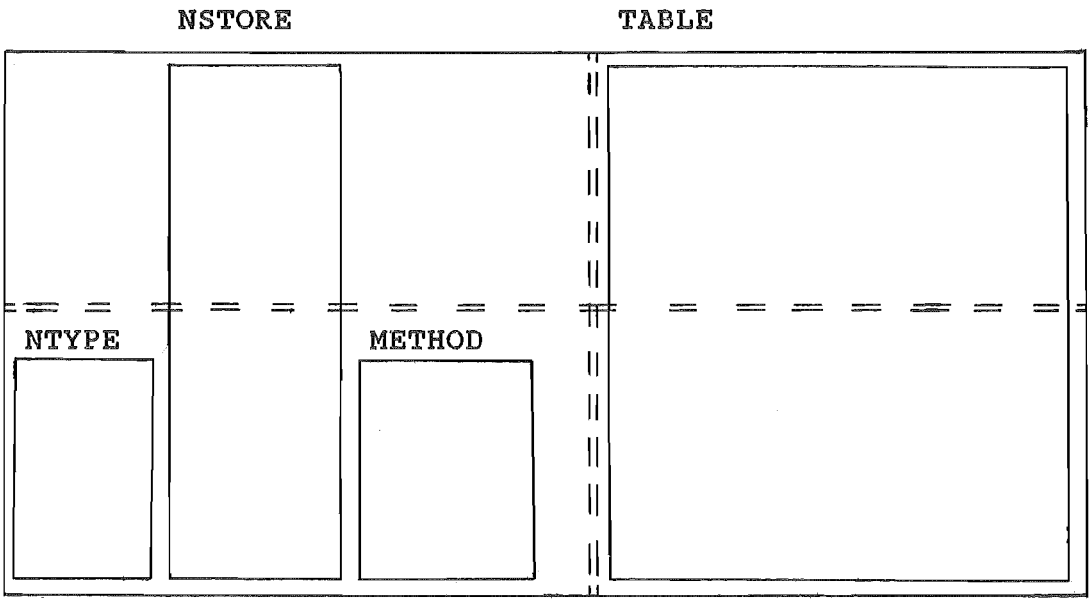


Figure 5.2 Detail of decision table storage.

location of any variable required (NSTORE). Actions are restricted to four types:

1. Calculation of a variable (NTYPE = 1). As the method of calculation of a variable may vary from one application to another, a variable cannot be retricted to association with only one equation (e.g. the equation for the bending moment in a simply supported beam is different from that for a built in beam). Array NSTORE holds a pointer indicating the storage location of the variable which is to be calculated (arrow 2, figure 5.1), while METHOD points to the equation to be used in the calculation (arrow 7, figure 5.1).

2. Displaying the value of a variable (NTYPE = 2). The storage location of the variable is indicated by NSTORE. No method is indicated as the action requires the current state of the variable to be displayed.

3. Direct execution of another decision table (NTYPE = 3, NSTORE = null). A pointer (METHOD) indicates the decision table which is to be executed (arrow 6, figure 5.1).

4. Displaying a message (NTYPE = 4, NSTORE = null). A pointer (METHOD) indicates the location in the message file of the message to be displayed. (arrow 8, figure 5.1).

These modifications to the condition and action stubs are made by the system programs after the decision table has been entered. Condition and action entries are stored in their original forms.

2. CASES

The values of data elements (including condition stub members) are stored in the array CASE. Storage locations start at address 51 as mathematical operations are numbered from 1 to 35 and look-up tables are numbered from 36 to 50. Associated with each storage location is a PResence of Data flag stored in NPRD.

Presence of data flags are normally boolean status indicators. The status of a data element is valid (NPRD = 1) if it has been calculated using the current value of each of its ingredients (i.e. presence of data flags for each of its ingredients are set to valid). The status of a data element is void (NPRD = 0) if it has not yet been calculated, or if a change to the value of one or more of the data elements in its global ingredience rendered the calculated value no longer valid.

There is one exception to the '0' or '1' status of the presence of data flags; this arises when the normally design program-evaluated value of a data element is overridden with a value preset by the designer. Thus, for the purpose of the next design iteration, the data element is effectively a designer-controlled variable with its presence of data flag set to valid. However, should the designer choose to subsequently alter the value of a true designer-controlled variable

which is a member of the overridden variable's global ingredience, any valid data elements (i.e. NPRD = 1) in the global dependence of the altered designer-controlled variable would, in the normal course of events, be set to void (NPRD = 0) thereby defeating the object of the override in the first place. Hence, when the value of a data element is overridden its presence of data flag is set to '2' (NPRD = 2), even though its status for use in the calculation of its dependents is effectively set to '1'.

Multiple sets of NPRD and CASE are stored, allowing a case study approach to the design process whereby the designer may use one of a number of sets of data. Storage locations are the same for each list so other information, such as the dependence list associated with each location, does not require multiplication. Different case studies can be reserved for specific purposes, such as standard base data, or for use by different system users or to record successive design iterations for future reference.

After an existing case study is selected, its contents are copied into case study 1, which is reserved as a work area, leaving the original data intact. Thus, the designer may commence work using the parameters of an existing design and produce a modified version of it. Should the designer wish to start from scratch, case study 1 is selected as a starting point. On reaching a satisfactory design the designer has the choice of whether and where to save the data currently in the work area. The design may be saved as a modification of the original data or saved saved in its own right in another area. Should the design session become non-productive the designer can reject the data in the work area

and start again with a fresh copy of the original data.

In this project the number of case studies has been arbitrarily set at six including the work area.

3. TRANSFORMATION AND DESCRIPTIVE DATA

When a data element is to be evaluated by execution of a decision table, the TABle Designation NTABD indicates the decision table to be used (arrow 4, figure 5.1). If a data element is to be calculated directly from its ingredients, the equation whose number is stored in NEQN is used (arrow 9, figure 5.1).

When the value of a data element is to be determined, NTABD is checked first and if it is non-zero the indicated table is executed. If no table is indicated and NEQN is non-zero, the indicated equation is used for direct calculation (arrow 9, figure 5.1). If both NTABD and NEQN are zero then the element is a designer-controlled variable whose value must be assigned directly by the designer.

In cases where both NTABD and NEQN are non-zero the indicated decision table is executed; the equation number is only significant during the input of the data network structure. This will be detailed in Section 4 of this chapter.

There are two sets of descriptive data associated with each storage location. NAME stores the variable name, which for a condition stub expression is the condition expression itself, and UNITS stores the

units in which the value of the variable is to be entered. Units are descriptive only and a unit compatibility check is not made by the system

4. INGREDIENCE LISTS AND EQUATIONS

Pure ingredience list are not stored or used by the system as they are implied in the equations supplied by the designer. Two forms of each equation are stored in CALC: the raw form, as entered by the designer, and its Reverse Polish form. The raw form is retained for the benefit of the designer in case it is required for viewing or editing. During processing of the raw equation, operators are replaced by a code number in the range 1 to 30, variable names are replaced by pointers indicating a storage location (arrow 3, figure 5.1) and look-up table names are replaced by a look-up table number in the range 36 to 50. Spare code number slots are available if future designers wish to insert custom operations. The first element of the processed list tells how many elements are in the list. When a look-up table number is encountered during execution, the table number gives access to the starting address in the storage array RLKDAT, and the table arguments combined with other data allow the actual data location to be identified.

To enable all the equations associated with any particular data element to be identified, the equations are linked using a linked list format with the first equation indicated by NEQN (arrow 9, figure 5.1) which, in turn, has an associated pointer indicating the location of the next equation (arrow 10, figure 5.1). These links are not used during

run-time when decision tables are used to decide which is the appropriate equation to use (arrow 7, figure 5.1).

The convention used in the Program Generator, when storing lists of data such as processed equations, is to use the first entry in the storage area to indicate how many entries are in the list.

5. LOOK-UP TABLES

Look-up tables provide a user-defined array-type data storage facility which is intended for use when the designer would otherwise have to refer to tables of standard or other reference data. During execution look-up tables are read only, their values having been entered by the designer during the program input phase. A total of 15 look-up tables may be created.

In the Reverse Polish form of an equation the pointers indicating the storage locations of the look-up table arguments are encountered first, followed by the number of the look-up table itself. Data for all look-up tables is stored in sequence in the array RLKDAT. LKPTR holds pointers indicating the starting position within RLKDAT of each look-up table (arrow 17, figure 5.1). LKDIM gives the number of axes of each table and is used during the running of a program to check that the correct number of arguments have been included in the equation concerned. Other storage areas are concerned with storing look-up table identification information and other parameters which are primarily used during the entry of look-up tables by the designer. The purposes of these storage areas are as follows:

LKNAME look-up table name

LKDIM number of axes or arguments associated with the table

LKSIZE number of entries along each axis of the table

LKAXIS name of each table axis

LKTOT total number of entries in each table

LKPTR pointer to the starting location in RLKDAT for each table

LKNEXT pointer to the next input slot in RLKDAT to be used by a new table

6. DESIGNER-CONTROLLED VARIABLE LISTS

Associated with each decision table is a list of designer-controlled variables, NDCV; the value of each of these variables may be changed before the decision table is run. The list format is the same as that for dependence lists, with pointers listed instead of data element names (arrow 12, figure 5.1); the first element in the list gives the list length.

The same designer-controlled variable will occur in more than one list if it is in the global dependence of more than one condition expression, but because NPRD is set to 'valid' after the first query of its value the designer will only be queried once per program run.

7. RESET LIST

When a designer-controlled variable is encountered during the

running of an application program, a pointer indicating its storage address in CASE is added to the reset list NRESET. This records that the variable's presence of data flag, NPRD, has been set to 'valid' after the designer was given the option of changing its value. At the completion of the run the presence of data flags at all addresses on the list are reset to 'void' (NPRD = 0), so that the system will give the designer the option of changing the value of the variable during the next program run, and the list length is zeroed.

Pointers indicating the storage locations of overridden variables are also stored in NRESET so that the variable is restored to its normal status after the completion of the design program run.

Unlike the other data structures which have been discussed, the state of NRESET is dependent on the running of the application program rather than on its generated structure. As the list is zeroed after each program run is completed it exists only in memory and is not permanently stored.

8. DIRECTORY OF DATA ELEMENTS

The data storage areas discussed thus far use storage addresses instead of variable names as a reference. The link between names and storage addresses is provided by the ordered directory DIRECT which documents data element names and their corresponding storage locations. The directory is ordered firstly on name length and secondly on alphabetical order. Names of up to 20 characters long are permitted. The first section of DIRECT points to the positions within the storage area

of the block corresponding to each name length as is illustrated by arrows 15 and 16 in figure 5.1. Arrow 14 in figure 5.1 illustrates the link between DIRECT and CASE.

9. DEPENDENCE LISTS

In figure 5.1, dependence lists are shown stored adjacent to the storage location of the data element to which they pertain. The dependence lists, instead of listing dependents names, contain pointers indicating the storage location of the element on the list (arrow 5, figure 5.1). The first element of the dependence list tells how many pointers follow in the list. Associated with each entry in a dependence list stored in NDEPN is an equation number stored in NSORCE. The equation identified in NSORCE is that which gave rise to the dependence. The NSORCE equation number is used by the WARN3 algorithm detailed in Chapter VII, to determine whether the associated dependence relation recorded by NDEPN applies at a particular time.

As was outlined in Chapter III, dependence relationships can arise from the decision tables themselves. A change in the value of a condition stub expression can change the rule which applies and, hence, the values of any relevant action stub expressions (see figure 3.3). To enable these relationships to be traced during the running of an application program, dependence lists are also used to record dependent decision tables. If it was not for the decision table, a condition expression would not have any dependents. Hence, if the first entry of the dependence list is zero, the associated data element is a condition expression. To preserve this characteristic and to enable a decision

table number to be distinguished from a variable storage address, lists of dependent decision tables, associated with condition expressions, start with a leading zero and the second entry in the storage area gives the list length, the remaining entries are the dependent table numbers. Thus, the global dependence of a data element can be traced, any dependent condition expressions detected and, through a knowledge of the last rule applying in the dependent decision table, any affected action expressions can be traced.

CHAPTER VI

GENERATING DECISION TABLES AND THE DATA NETWORK

The suite of algorithms associated with the program generator can be divided into two functional groups: the first for design program generation; the second for running the generated program. A third function, that of modifying the generated program, selectively uses all or part of the algorithms developed for program generation and so will not be considered separately.

The purpose of this chapter is to discuss the algorithms developed for design program generation.

1. DESIGN PROGRAM INPUT

The parent algorithm for program input is DESPGM, illustrated in figure 6.1. Execution of the algorithm is triggered when the design selects the 'enter a new program' option within the calling algorithm which then increments the number of decision tables by one and calls the DESPGM routine. Execution is repeated until the new decision table, and any subsequent decision tables and all resulting variables and look-up tables have been entered. The algorithm is structured so that all variables downstream from the first decision table are processed before processing of the next decision table commences. This is in line with the top-down logic of the problem itself. Thus, the first decision table a designer enters is the one which makes the top-level decision, for example 'calculated stress < allowable stress'. From this point onwards

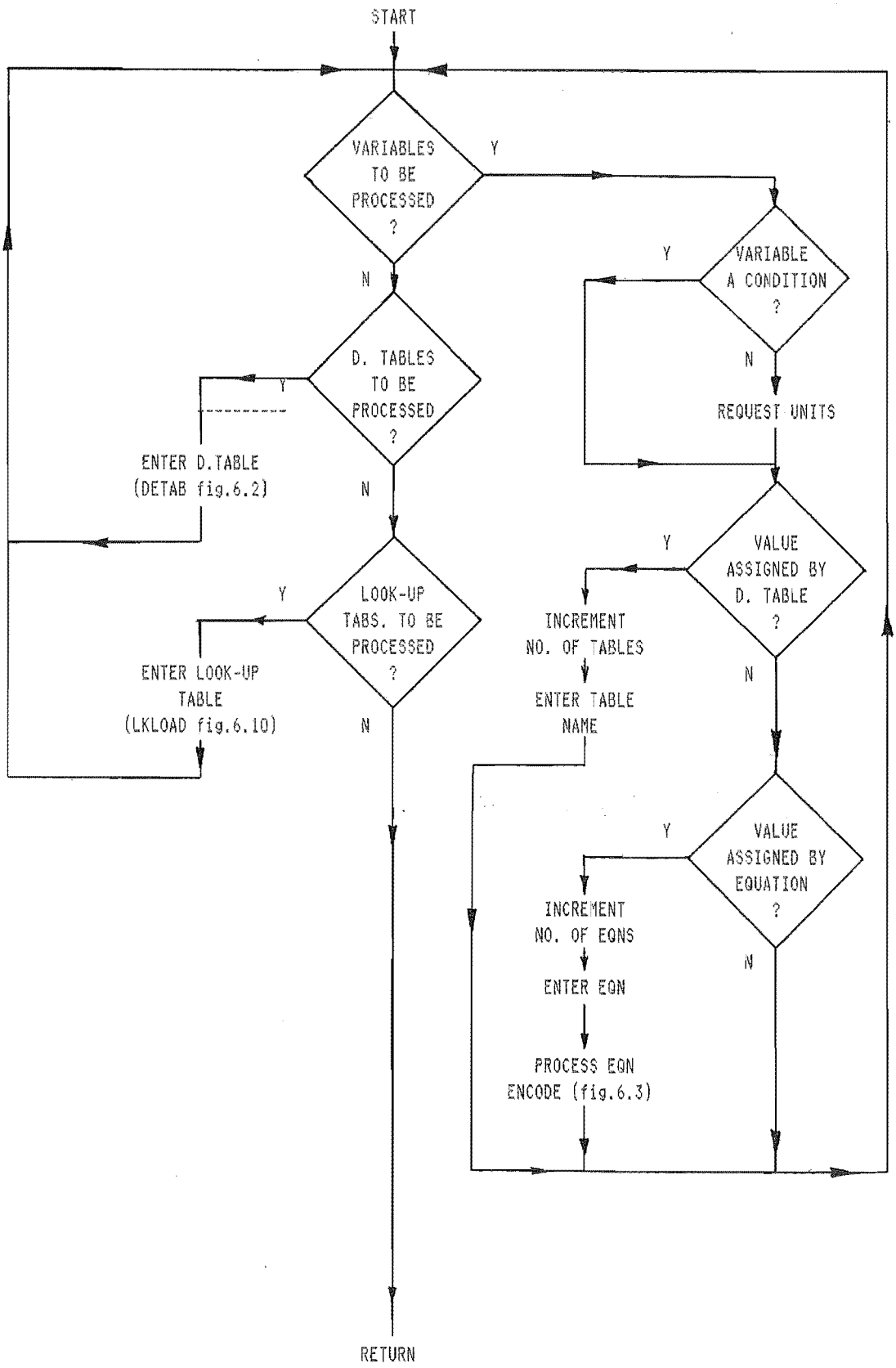


Figure 6.1 Algorithm DESPGM

it is the system which decides the order in which decision tables and equations are entered and the designer is prompted accordingly. This is made possible by the processing strategies adopted and the resultant data storage methods. The data input process can be viewed as a series of partial traversals of the developing data network from right to left, with each traversal being completed when all the data downstream from each decision table is allocated storage.

2. DECISION TABLE INPUT

Figure 6.2 represents the DETAB algorithm for decision table input. Parts of the subroutine built on this algorithm are also used when additions or alterations to decision tables are required.

As part of the processing of condition stubs, dependence relationships between decision tables and conditions are noted by recording the decision table number in the dependence list of each condition stub expression. This information allows potentially affected decision tables to be identified when, during the running of a design program, changing the value of a variable affects all elements in its global dependence.

During processing of actions which call for calculation of a variable, the algorithm calls for entry of units for the variable and an equation for use in calculation of the variable. This may, at first, appear to duplicate parts of the DESPGM algorithm of figure 6.1, however, within DESPGM the variable is calculated directly from its ingredients whereas within DETAB provision must be made for selecting

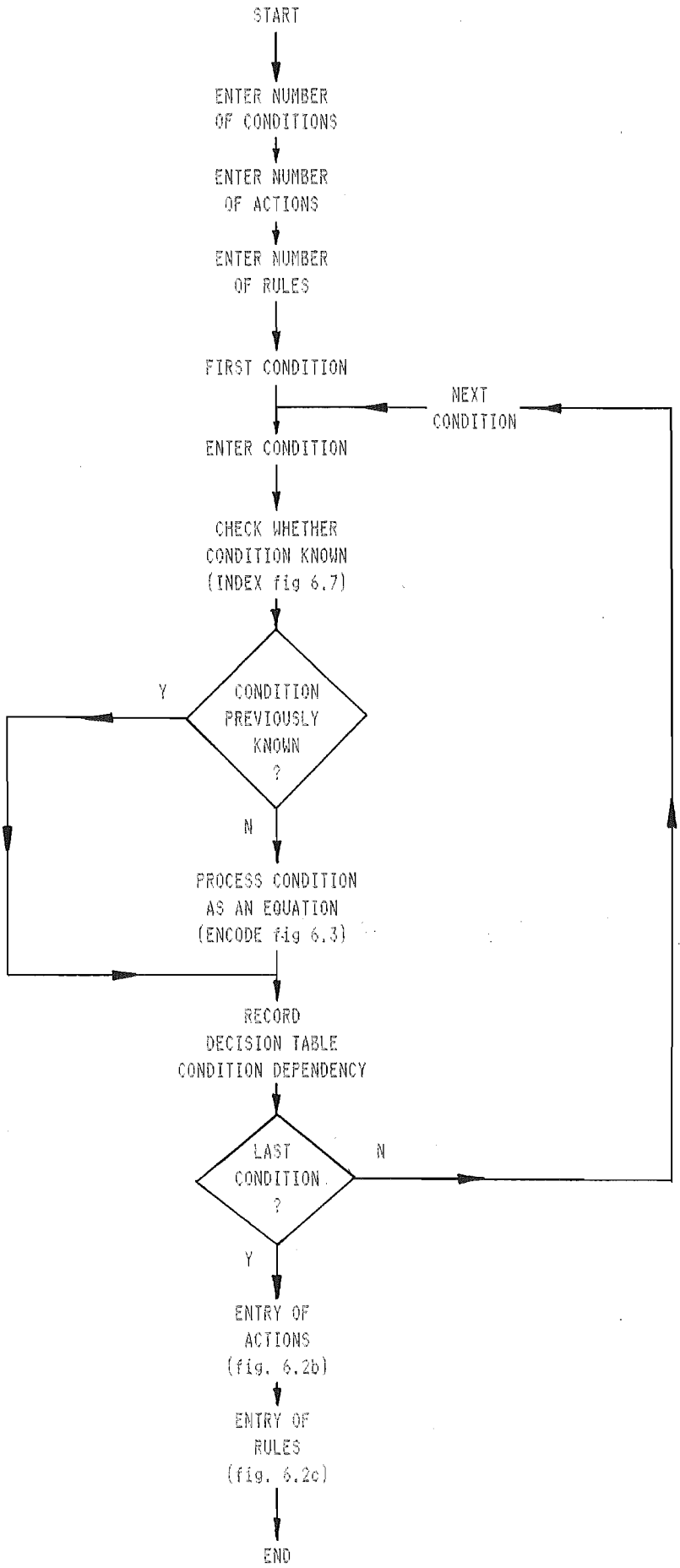


Figure 6.2a Algorithm DETAB

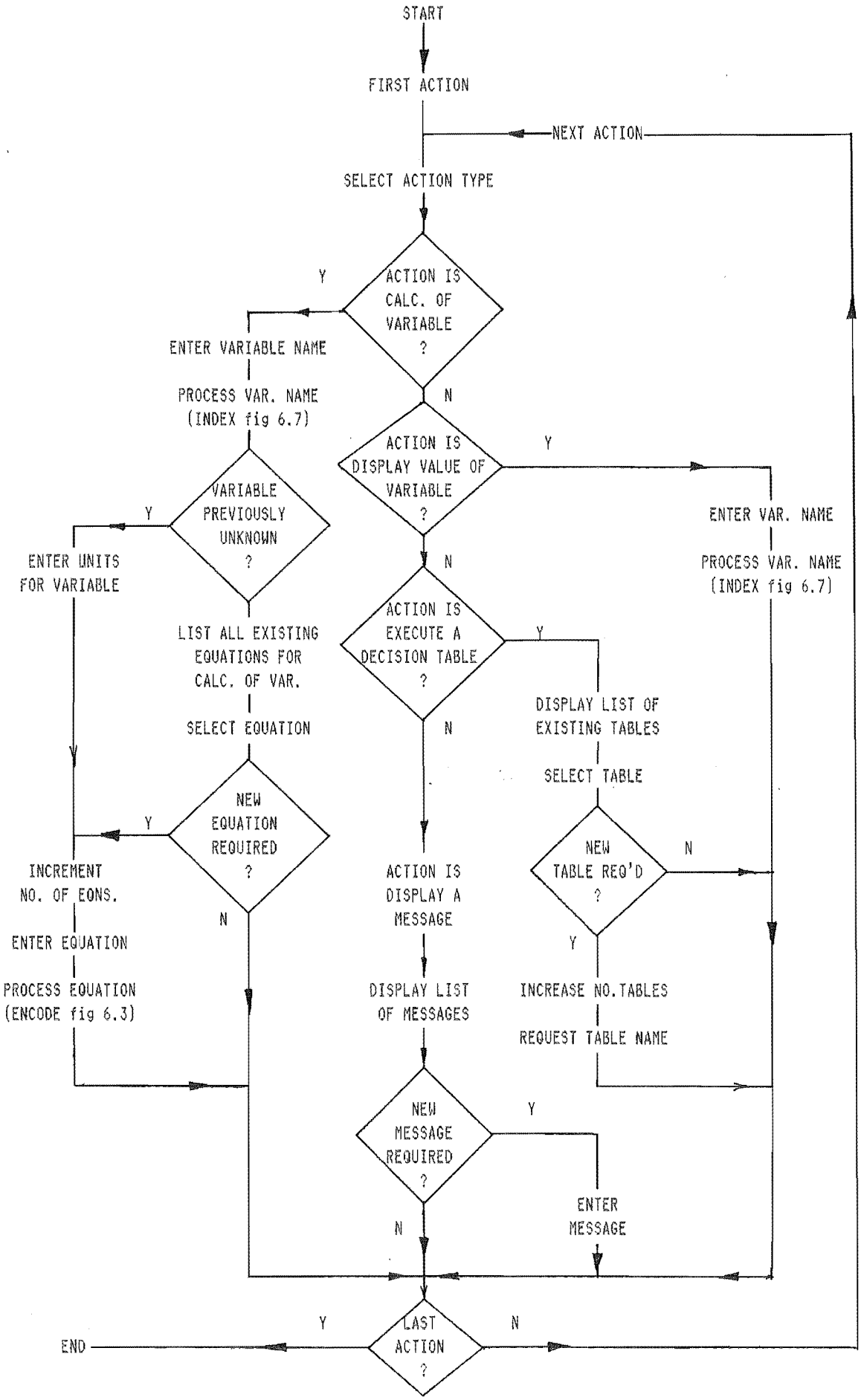


Figure 6.2b Algorithm DETAB. Entry of Actions

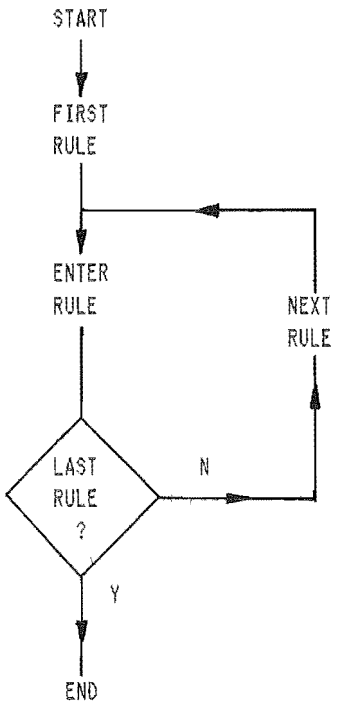


Figure 6.2c Algorithm DETAB. Entry of Rules

from a range of equations, the choice depending on the conditions which will prevail at the time of its use. Hence, the equation must be identified at this point and units are recorded at the same time.

3. EQUATION ENCODING

As an input equation is scanned using the ENCODE algorithm of figure 6.3, a second (temporary) equation is built up of operator codes, variable storage addresses and look-up table numbers instead of the names and symbols used originally. A list of codes is shown in figure 6.4. The key part of the algorithm is the use of a list of standard keywords such as operators ('+', '-', etc.), trigonometric and logorithmic functions ('sin(', 'log(' etc.) and digits ('0' to '9'). The use of various checks allows identification of the embedded variable and look-up table names. If pi, yes, no, true or false are identified as keywords, a check is made whether it should be identified as such or if it is part of a variable or look-up table name.

Unlike a programming languages such as FORTRAN, the ENCODE algorithm allows embedded blanks within look-up table and variable names. This eliminates the need to eliminate the need to link separate words in one name, for example by underscoring, thereby eliminating a potential source of error for a novice user.

Neither leading digits in names nor numbers are allowed within equations. Instead of using a number the designer must use a variable to which the appropriate value is assigned during the running of the program. So that the variable can be treated as a constant, the

end of equation	Y	N	N	N	N	N	N	N	N	N	N	N	N
equation element = blank	-	Y	N	N	N	N	N	N	N	N	N	N	N
keyword encountered	-	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
keyword part of a name	-	-	Y	Y	N	N	N	N	N	N	N	-	-
equation element='('	-	-	-	-	N	Y	N	N	N	N	N	-	-
equation element=')'	-	-	-	-	-	-	Y	Y	Y	N	N	-	-
scanning look-up arguments	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-
new look-up table	-	-	-	-	-	-	Y	N	N	-	-	-	-
old look-up table	-	-	-	-	-	-	N	Y	Y	-	-	-	-
no. look-up arguments correct	-	-	-	-	-	-	-	N	Y	-	-	-	-
last equation element a letter	-	-	-	-	Y	Y	Y	-	-	Y	Y	N	-
this & last element compatible	-	-	Y	N	-	-	-	-	-	-	-	Y	N
equation element=comma	-	-	-	-	-	-	-	-	-	Y	N	N	-
equation element=digit	-	-	-	-	-	-	-	-	-	Y	Y	-	-
ERROR			X			X			X		X		X
(a name is being scanned)		X	X					X		X		X	X
(keyword marks end of name)				X									
('('marks end of look-up name)					X								
(')' marks end of look-up arguments)						X		X					
(leading digit encountered)											X		
INDEX name (fig.6.7)				X	X								
enter keyword code into temp. eqn.				X	X								
enter var. address into temp. eqn.				X		X		X					
enter look-up no. into temp. eqn.					X								
set parameters for look-up table						X							
increment no. look-up table arguments									X				
write original equation to file	X												
form equation linked list	X												
convert to RPN (fig.6.4)	X												
update dependence lists (fig.6.9)	X												
increment pointer		X	X		X	X	X		X	X	X		X	
re-execute this table		X	X		X	X	X		X	X	X		X	
exit from table	X			X				X				X		X

Figure 6.3 Logic of ENCODE algorithm

Symbol	Code	Symbol	Code
=	1	LN(	18
<	2	LOG(	19
>	3	PI	20
,	4	0	21
-	5	1	21
+	6	2	21
**	9	3	21
^	9	4	21
*	7	5	21
/	8	6	21
(	10	7	21
)	11	8	21
SIN(	12	9	21
COS(	13	TRUE	22
TAN(	14	FALSE	23
ASIN(	15	YES	22
ACOS(	16	NO	23
ATAN(	17		

Note: during equation encoding other characters use code 33

Figure 6.4 Symbols and their codes

automatic querying of designer-controlled variables during each run may be suppressed. This strategy was adopted to avoid the complications of having to assign a numerical value to a character substring consisting of digits, which may have been expressed in different but numerically equivalent forms.

A checking matrix is used to check the compatibility of adjacent equation elements, e.g. a variable may not follow a close bracket ')'.

4. CONVERSION TO REVERSE POLISH NOTATION

The ENCODE algorithm encodes the raw equation into a numerical form with elements in the same order as the original. The RPN algorithm of figure 6.5 retains the numerical form but reorders the equation into Reverse Polish notation. Some sample conversions using RPN are shown in figure 6.6 which retains the original symbols for simplicity.

The RPN algorithm is based on a scanning of the preprocessed equation using up to two active pointers at any one time. Pointer 2 is the leading pointer and is incremented along the preprocessed equation, whereas pointer 1 is a marker to be inserted where and when required. Variable addresses are put into the Reverse Polish equation as they are encountered by pointer 2. If pointer 2 encounters an operator, pointer 1 is inserted, and the incrementing of pointer 2 continues. If subsequently another operator is encountered by pointer 2, if the operator at pointer 1 takes precedence, as in figure 6.6a, it is entered into the new equation. Pointer 1 is then moved to the position indicated by pointer 2, which continues to be stepped along the equation. If, as

pointer 2 at variable address	Y	N	N	N	N	N	N	N	N	N	N	N
pointer 2 at end of equation	-	Y	Y	N	N	N	N	N	N	N	N	N
pointer 2 at right bracket ')'	-	-	-	Y	Y	Y	Y	N	N	N	N	N
pointer 1 active at this level	-	Y	Y	Y	N	Y	N	-	N	Y	Y	Y	Y
level=1	-	Y	N	N	N	Y	Y	-	-	-	-	-	-
pointer 2 at operator	-	-	-	-	-	-	-	Y	Y	Y	Y	Y	Y
pointer 2 at comma	-	-	-	-	-	-	-	Y	N	N	N	N	N
operation at pointer2 takes precedence	-	-	-	-	-	-	-	-	-	N	Y	Y	Y
pointer 1 at - & pointer 2 at +	-	-	-	-	-	-	-	-	-	-	-	Y	-
pointer 1 at * & pointer 2 at /	-	-	-	-	-	-	-	-	-	-	-	-	Y
put element at pointer 2 into RPN eqn	X											
ptr1 (this level) element into RPN eqn		X		X		X				X		X	X
deactivate pointer 1 at this level		X		X	X	X	X			X		X	X
stack, increment level											X		
activate pointer1, pointer1=pointer2									X	X	X	X	X
unstack, decrement level			X	X	X	X	X			X		X	X
increment pointer 2	X				X	X	X	X	X	X	X	X	X
re-execute this table	X		X	X	X	X	X	X	X	X	X	X	X
exit this table	X												

Note: pointer 1 exists at different levels &
references to pointer 1 apply to the current level.

Figure 6.5 Logic of RPN algorithm

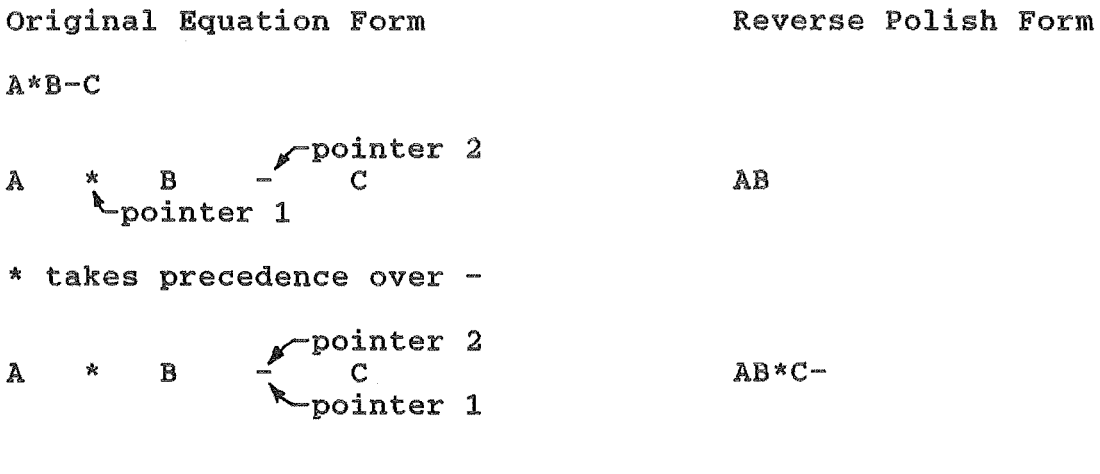


Figure 6.6a Sample equation conversion.

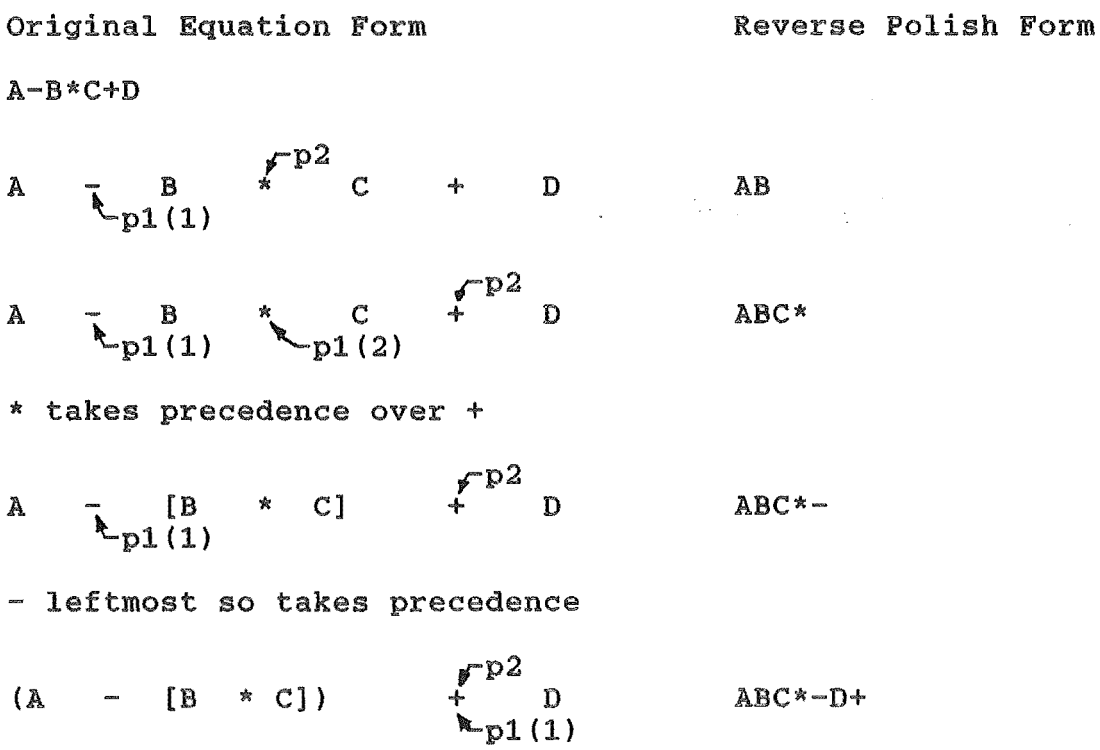


Figure 6.6b Sample equation conversion.

in figure 6.6b, the operator at pointer 2 takes precedence, it cannot immediately be entered into the equation in case there is a third operator which takes precedence over the operator at pointer 2. To allow this, stacking occurs: the position of pointer 1 is stored at 'level 1' and the working level is increased from 1 to 2. Processing continues ignoring pointer 1 (at level 1), as if the operator at pointer 2 was the first operator encountered. A new marker, pointer 1 (at level 2) is then inserted at the position of pointer 2 and scanning continues. When the + (addition) is encountered, * (multiplication) takes precedence and is entered into the equation. Unstacking then occurs: pointer 1 (at level 2) is deactivated, and operation resumes at level 1 where the original pointer 1 is reactivated. A comparison is then made between + at pointer 2 and - (subtraction) at the reactivated pointer 1 (at level 1). Addition has a higher valued code but a separate check for equal precedence operators occurring at the two active pointers (i.e. - or * at pointer 1 corresponding to + or / respectively at pointer 2) enters the left-most of the two operators, -, into the equation, followed by the addition operation when pointer 2 encounters the end of the equation. Thus, only one pass of the preprocessed equation is required to build up the new equation in Reverse Polish form.

5. INDEXING OF DATA ELEMENTS

The directory DIRECT is accessed by the INDEX algorithm which conducts a linear search of the appropriate name-length block to check whether the variable name given to it by the calling algorithm exists. If the name is found the corresponding storage location in CASE is returned. If INDEX finds that a particular name does not exist in the

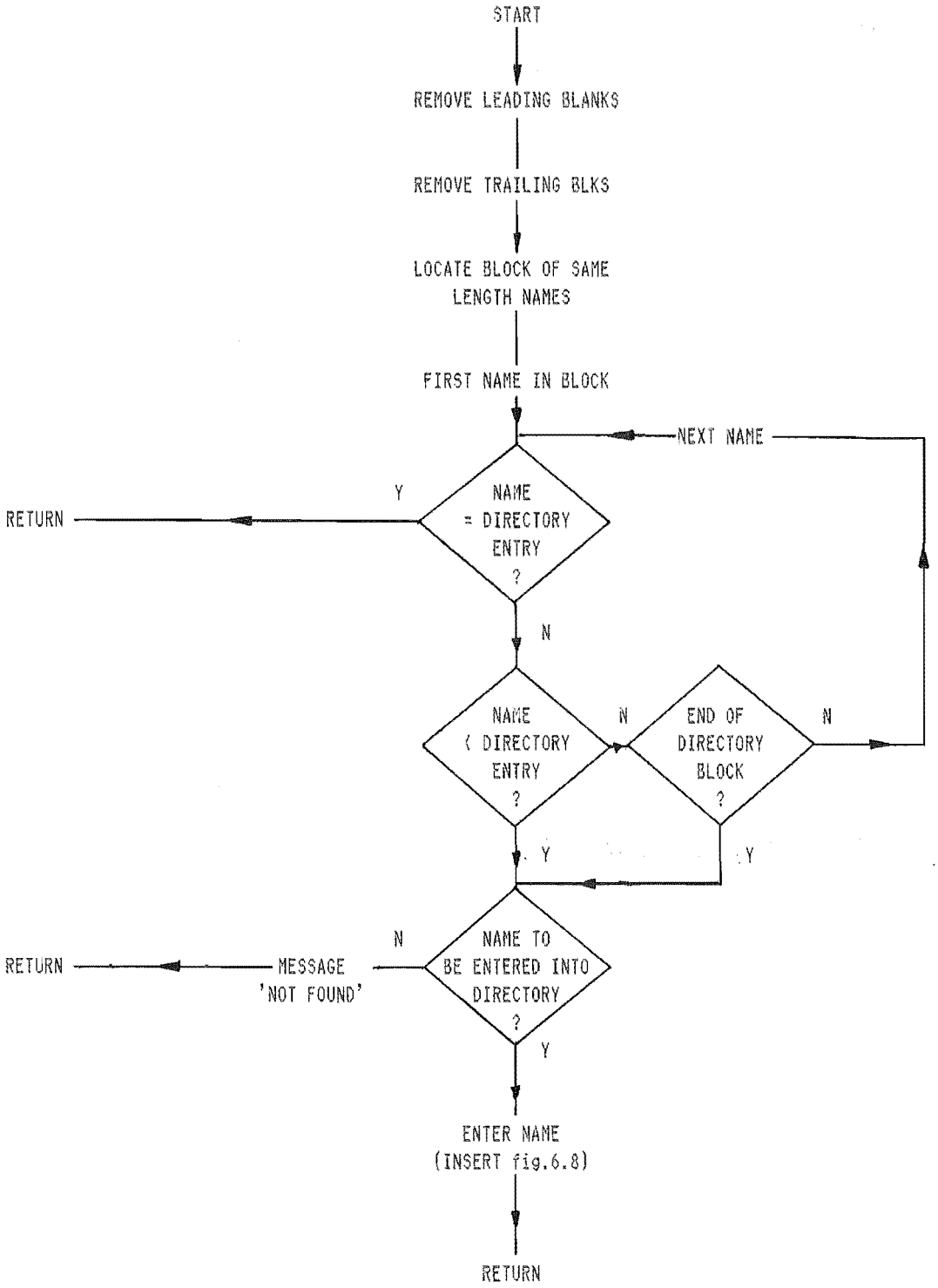


Figure 6.7 Algorithm INDEX

directory the algorithm can be used to mark the position where the name would be stored if it were to be entered into the directory, for instance during the scanning of an equation.

6. ENTERING DATA ELEMENTS INTO THE DIRECTORY

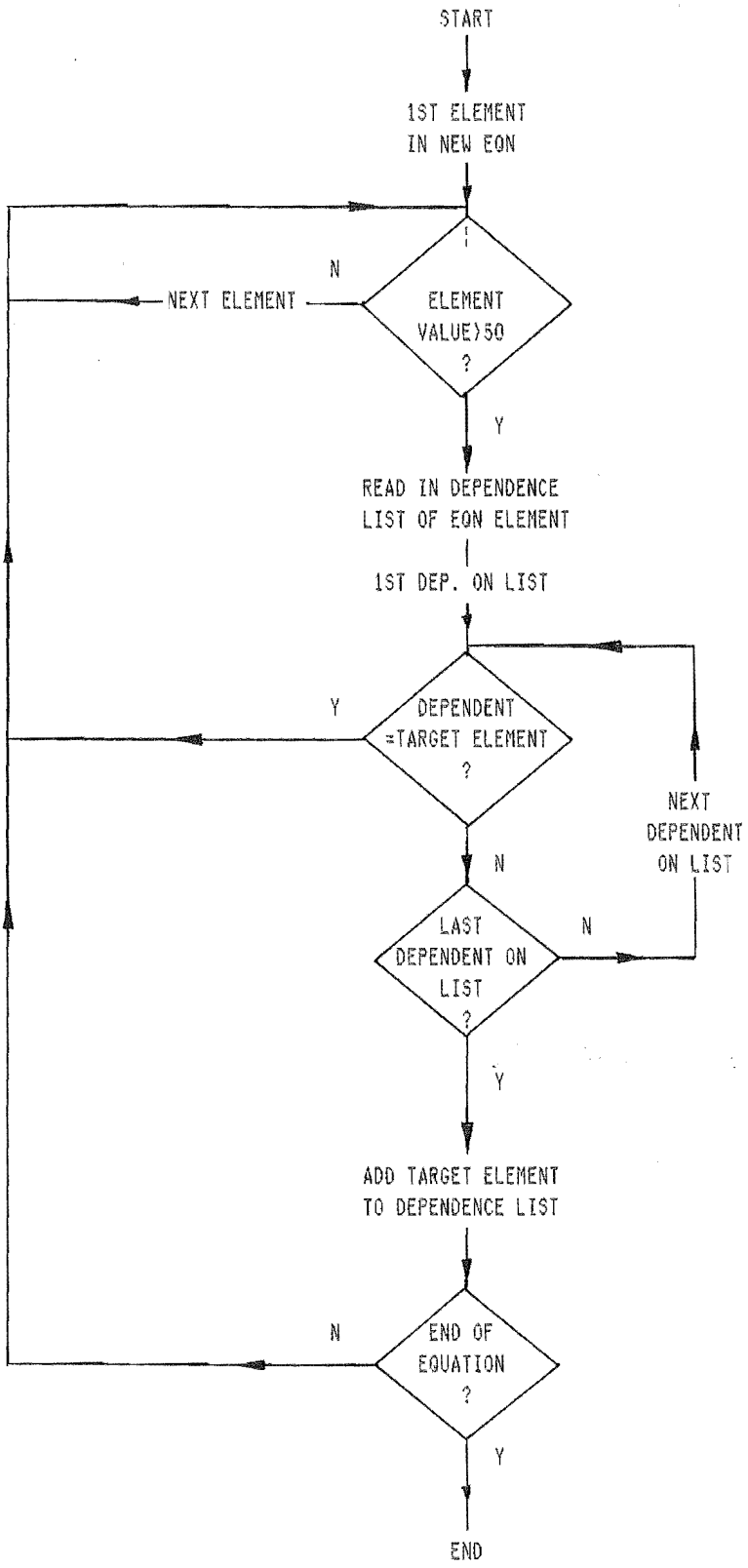
When INDEX determines that a new variable name is to be added to the system, the location which the new data is to occupy in the directory is passed to the INSERT algorithm (figure 6.8) which manipulates the data in DIRECT. If the space between the target block (the block corresponding to the length of the new name) and the following block is insufficient, the algorithm checks the space between all adjacent blocks and moves them within the storage area so that there will be five clear spaces between blocks after the new name is inserted. This reduces the frequency of directory data shifts.

7. MAINTENANCE OF DEPENDENCE LISTS

After an equation has been encoded (using ENCODE) and converted to Reverse Polish notation (using RPN) it is processed by the algorithm DEPLST which maintains dependence lists. Feeding DEPLST an equation gives it the same information as would feeding it an ingredience list. To maintain dependence lists, the subject of the equation needs to be added to the dependence list of every one of its ingredients. Hence, DEPLST consists of two nested loops: the outer to scan the equation for variable addresses and the inner to scan the dependence list of each variable address in the equation and add the target variable address to the list if it does not already feature.

space at end of name-length block)1 new name to be at end of block	Y Y N N N Y N Y
calc shift to give each block 5 spaces shift blocks to give 5 clear spaces increment no. variables in system move data in block to make space insert name as indicated by INDEX update record of block positions exit this table	X X X X X X X X X X X X X X X X X X X X X X

Figure 6.8 Logic of INSERT algorithm



Note: the 'target element' is the LHS of the new equation

Figure 6.9 Algorithm DEPLST

When equations are modified they are reprocessed by ENCODE, and DEPLST is then re-run. If any variables have been removed from the original equation a routine UNDEPN removes the subject of the equation from the dependence list of the ex-ingredient.

8. LOOK-UP TABLE LOADING

Algorithm LKLOAD, illustrated in figure 6.10, interactively loads the designer's tabulated look-up data into the storage area RLKDAT. Up to five table axes can be accomodated and, after loading the axes' descriptions and lengths, five nested loops control the loading of data elements into successive slots in RLKDAT.

9. MODIFYING A PROGRAM

A generated program can be readily edited or expanded by a 'modify' option which makes selective use of the algorithms discussed above.

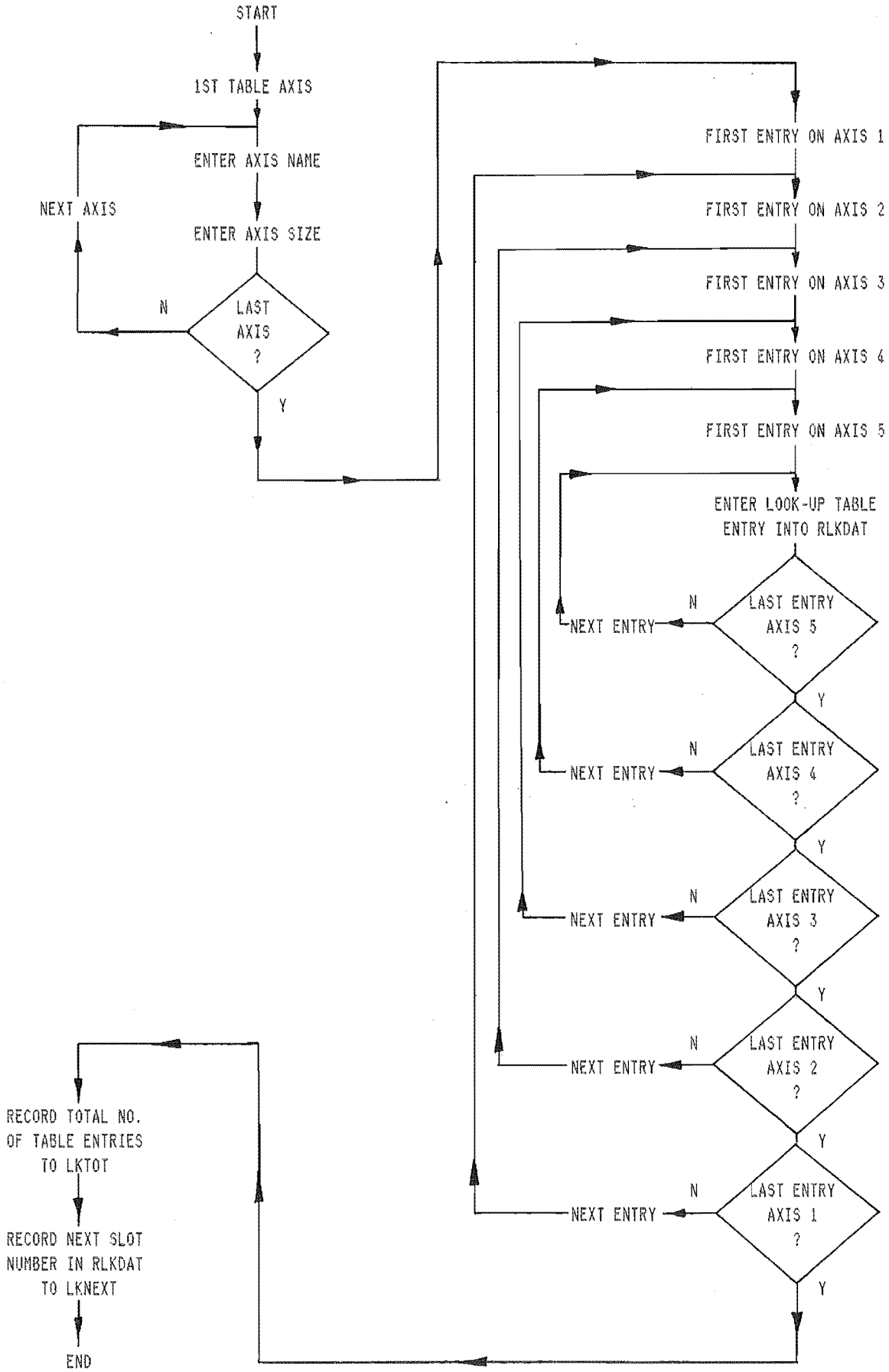


Figure 6.10 Algorithm LKLOAD

CHAPTER VII

DECISION TABLE EXECUTION AND DATA PROCESSING

The algorithms associated with running a generated application program fall into two groups: those which actually run the program, and those which provide optional data manipulation or information on the progress of the design or the program structure. The first group contains the key algorithms TABEX3 and WARN3 which run the decision tables and provide the data change and updating facilities. The second group of algorithms are used between runs to override the values of normally calculated variables, suppress the once-per-run querying of designer-controlled variable values, move case study data to and from the work area, view decision tables and other program structural information, and obtain current values of variables and look-up table entries. All options are menu-driven from DESRUN, the run-time equivalent of DESPGM.

1. RECURSIVE EXECUTION

Execution of a decision table involves a left-to-right scanning of the condition entry to find the appropriate rule, and to perform the indicated actions. To enable this process to take place one or more of the the condition expressions will need to be evaluated; this will possibly involve the global dependencies of the expressions. Hence, the algorithm developed for processing decision tables, TABEX3 (figure 7.1a), has two paths: one for data evaluation (figure 7.1b), the other for decision table execution (figure 7.1c). These form the two branches

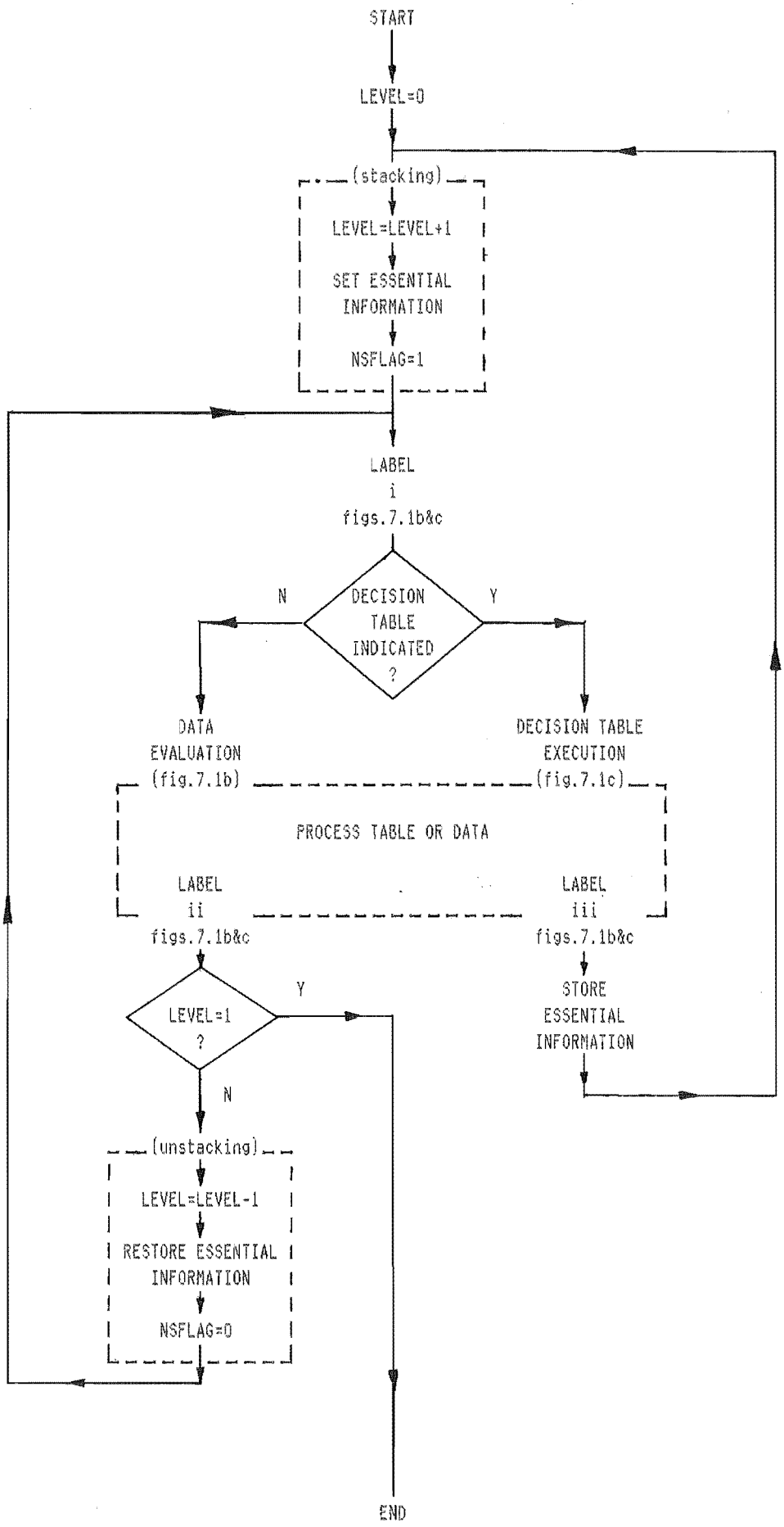
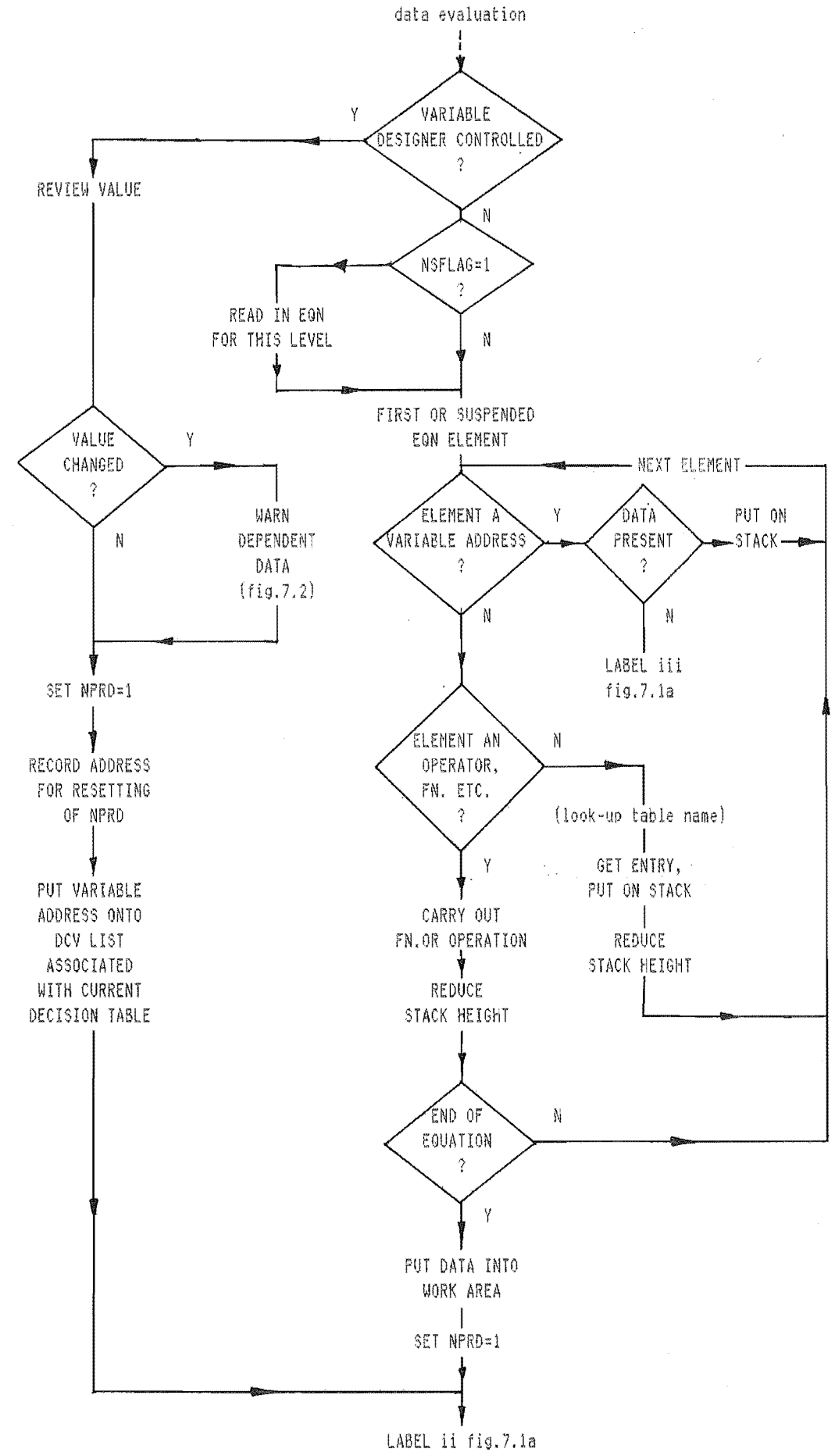


Figure 7.1a Algorithm TABEX3 (recursive processing)



Note: DCV=designer-controlled variable

Figure 7.1b Processing of data

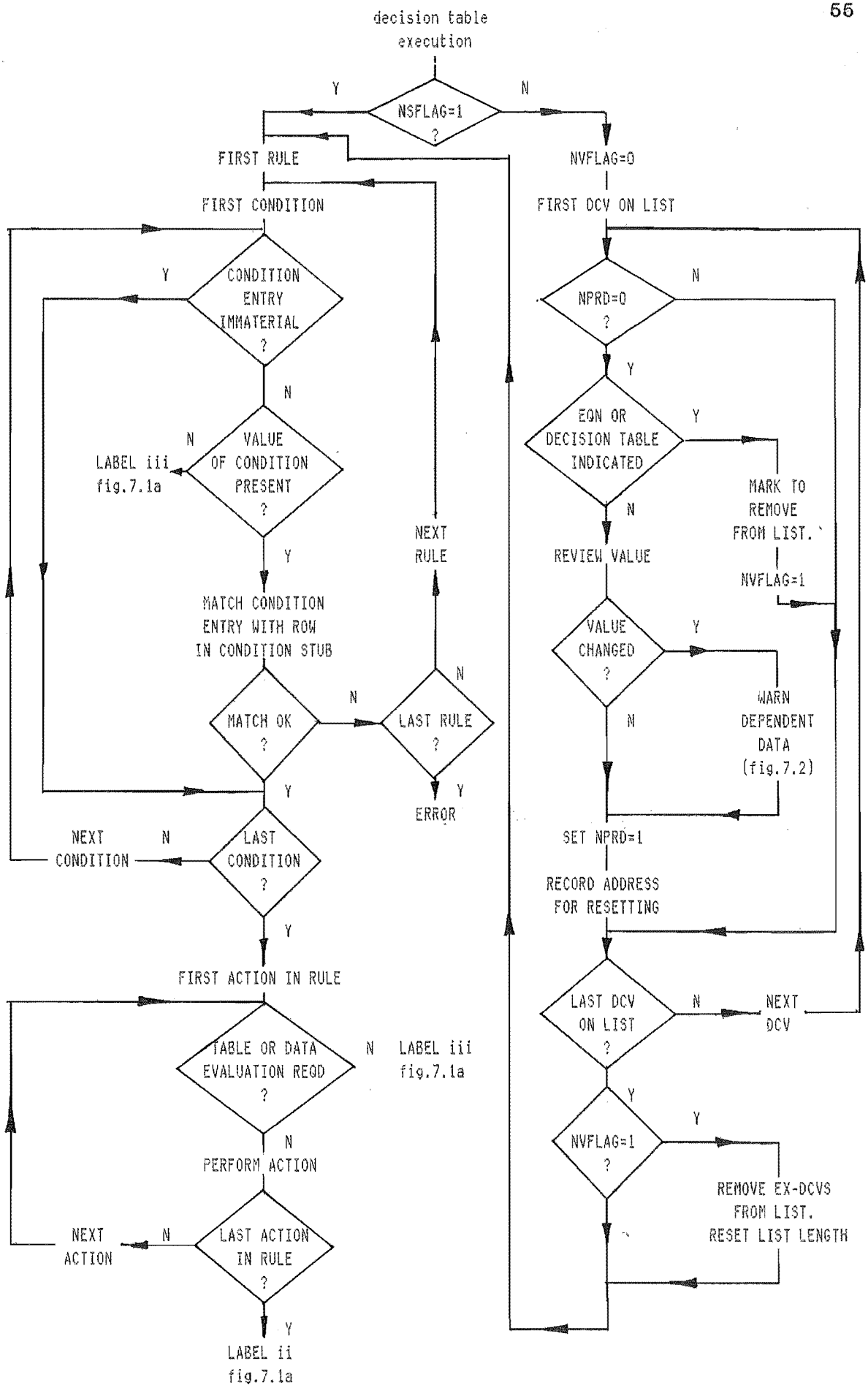


Figure 7.1c Decision table execution

of figure 7.1a as indicated. The basic principle of recursive descent and the rule scanning process as used by Goel and Fenves (1971) were employed in the TABEX3 algorithm.

(1) Decision Table Execution

The basis of actual decision table processing, the left-hand path of figure 7.1c, is two nested loops: the outer steps through the rules in a decision table; the inner steps through the conditions in a rule in an attempt to successfully match every condition in a rule to the state of the condition stub.

If a condition entry is immaterial (-), the matching attempt is omitted and the matching process moves to the next entry in the rule. If a mismatch is found the rule is rejected without processing any further entries and processing restarts with the first condition of the next rule. If execution leaves the condition loop normally (through matching the rule with the state of the condition expression), the appropriate rule has been found. If execution leaves the rule loop normally (through there being no matching rule), an error condition exists; this might arise if the table represents an incomplete specification or if data values exceed intended bounds.

It is possible to derive a number of different flow charts from one decision table; each flow chart reflects a different ordering of the decisions listed in the condition stub of the decision table. The most desirable ordering of decision evaluations is the one requiring the least number of decisions to be evaluated before output is reached. Rules and conditions can be rearranged without affecting the logic

expressed in a decision table. An organised arrangement of conditions and rules can increase decision table processing speed. Possible ordering tactics are discussed by Humby (1973). In the present project the arrangement of rules and conditions is entirely the prerogative of the designer.

The TABEX3 algorithm uses a flag (NSFLAG) to indicate whether the current table is being encountered for the first time. Before processing of the new table starts the designer is presented with the value of each of the the designer-controlled variables on the list associated with the table and is given the option of retaining the current value or entering a new value (right side of figure 7.1c). If the value of a designer-controlled variable alters, all the data in its global dependence (which will probably include decision table condition expressions) is set to void and will be recalculated later, if required. If a designer-controlled variable has been queried its data presence flag (NPRD) is set to 1 (valid) so that it will not be queried again when other tables are processed during the design iteration. The addresses of all queried variables are added to the reset list to be reset to void (NPRD = 0) at the end of the iteration. Before giving the designer the option of changing the value of a listed variable, a check is made to determine whether the variable is still under the direct control of the designer. A change in status from being designer-controlled results if the variable has been edited since the table with which the list is associated was last run. When a change is noted, the address of the variable is removed from the list.

Designer-controlled variables must be reviewed before beginning

execution of the table, so that any dependent data is invalidated before an attempt is made to access that data during decision table processing.

(2) Data Evaluation

Before attempting to calculate a variable whose value is undefined a check is made to ascertain whether the variable is under the direct control of the designer; if this is the case the left-hand path of figure 7.1b is followed. After the designer has been given the opportunity to change the value of the variable, its storage address is added to the designer-controlled variable list associated with the current decision table. Once on the list, future querying of the value of the variable will be performed before execution of the table begins. All designer-controlled variable lists are built up in this way.

The right-hand path of figure 7.1b is followed when calculation of a variable is required. At the commencement of calculation of the variable, as indicated by the stacking flag (NSFLAG=1), the indicated equation is read in from file. The equation is used in combination with a push-down stack, which holds ingredient and intermediate data values, to calculate the value of the variable which is then transferred to the work area and the status flag of the variable is set to valid.

When a decision table is executed, the two branches of the TABEX3 routine (figures 7.1b&c) are used recursively to evaluate only the decision tables and data items needed, either directly or indirectly, in the execution of the table. Presence of data indicators (NPRD) play a key role in this conditional execution strategy by identifying whether

data is ready for immediate use or whether further evaluation is required. If the status of a particular data item is found to be invalid (NPRD=0) the current operation is temporarily suspended, the essential information which identifies what was being processed and how far processing had progressed at the current level is stored on a push-down stack, the level of the stack is increased (hence the term stacking) and evaluation of the missing data element or indicated decision table is begun at the new level, beginning at point (i) on figure 7.1a. Several stacking operations may be carried out in succession as ingredients of ingredients are found to require evaluation. Stacking may occur if a needed data element is undefined or if a decision table action calls for direct execution of a decision table or calculation of a variable. When evaluation of a data element is completed its value is stored, its data presence flag is set to valid, and control passes to point (ii) on figure 7.1a for unstacking. The stack level is reduced by one and the essential information which was stored at this level before stacking took place is used to restore the state of algorithm to that it had been at the point of suspension, and processing is resumed.

Thus, only one general purpose algorithm used recursively is required for both data evaluation and decision table execution, regardless of the nature of the particular application program generated by the designer.

2. RECURSIVE UPDATING

Several iterations of the design process are normally required before a satisfactory solution to the problem is reached, and even then

the designer is likely to test a number of alternatives before deciding on the most attractive solution. This development process involves changes to input data and hence derived values. Three possible strategies for coping with the data change situation were:

(a) to set the status of all derived data to void and recalculate data as required from the input data

(b) to set the status of all derived data to void and simultaneously replace the data with a recalculated value

(c) to invalidate only data within the global dependence of the modified inputs.

Options (a) and (b) are wasteful of computing resources as they invalidate valid data which may then have to be recalculated. The third option presents the most efficient approach because only affected data is invalidated and TABEX3 will only recalculate data that is required. This strategy is embodied in another recursive algorithm WARN3. Fenves (1972) and Goel and Fenves (1971) have used this method for processing only dependents of data, whereas WARN3 also traces dependent tables and trans-table dependences (dependent actions). The WARN3 algorithm is shown in figure 7.2.

Clearing of dependent data begins with the dependents of the altered data item. When one of the items has dependents, a stacking operation similar to that used by TABEX3 is used to suspend processing of the current list, store the essential information detailing the nature of the list (whether it belongs to a condition expression), and begin processing of the second list at a higher level. Stacking may

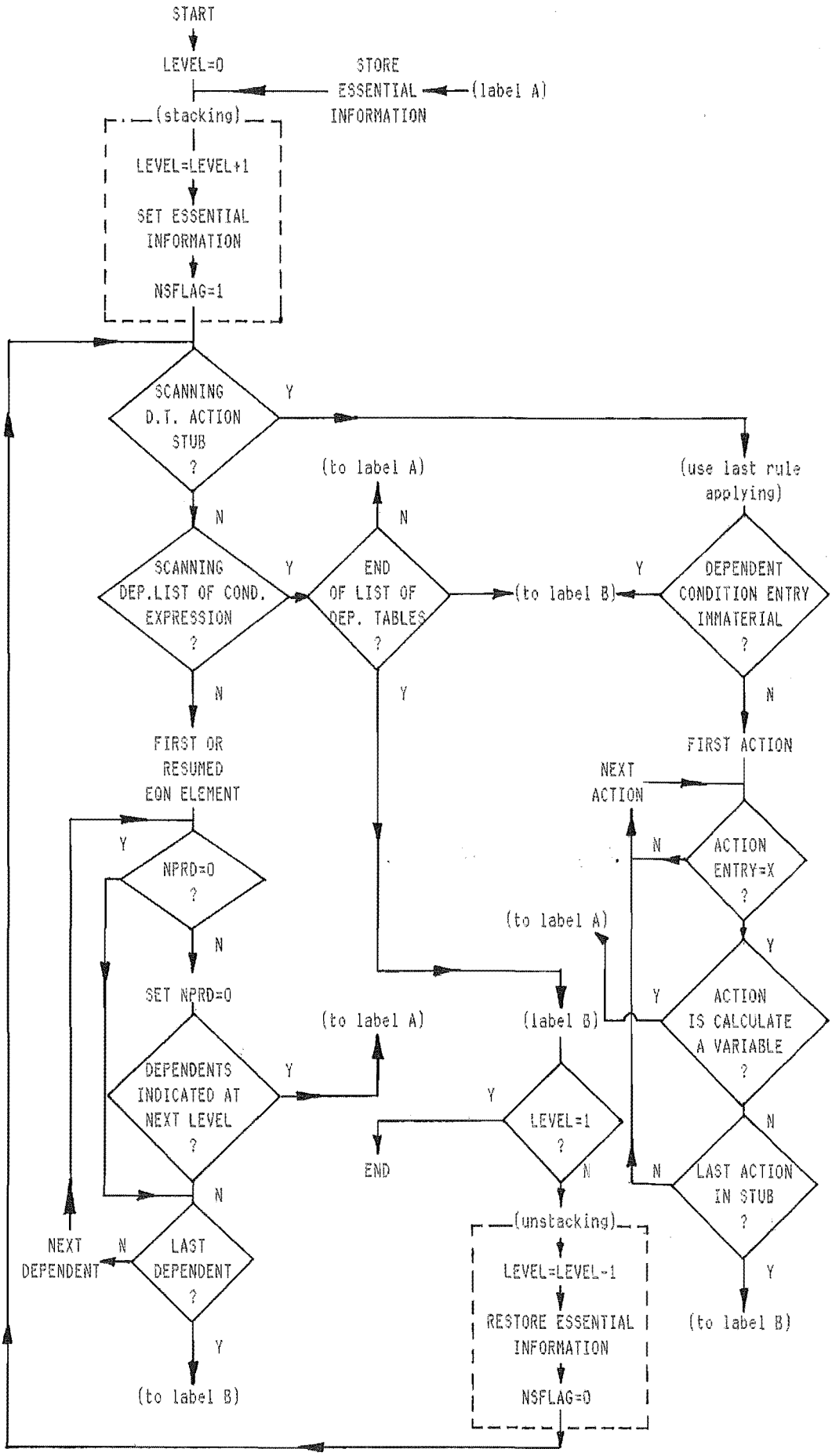


Figure 7.2 WARN3 (clearing of dependent data)

occur several times in succession as the global dependence of the altered data item is traced out. In due course the algorithm is likely to encounter a dependence list associated with a condition expression which contains the dependent decision table numbers. In contrast to normal data relationships, trans-table dependencies (refer figure 3.3) are not constant and depend on the state of the condition expression of the decision table. Once a condition expression is identified by WARN3, a check is made to establish whether the condition entry in the rule which applied during the last run of the table is - (immaterial); in such a case WARN3 does not proceed at this level because the state of the condition expression had no bearing on which rule applied when the table was last run. If the particular condition expression is relevant to the rule, then the action entry column corresponding to the last rule which applied is scanned to detect any actions which entail calculation of a variable. If any are found their presence of data flags (NPRD) are set to void and any dependents are, in turn, cleared.

Changing the value of a data element may invalidate a logical condition expression, but subsequent re-evaluation of the condition expression may well leave the state of the condition stub unchanged so that the same rule applies and the same actions are carried out. In such cases it is wasteful to clear action data and their dependents because of the assumption that the state of the condition expression had changed the rule which applied. Nevertheless, this option was used because immediate re-evaluation of the condition expression is probably more wasteful of resources as was outlined earlier this section.

3. DESIGNER-CONTROLLED VARIABLES

Specific provision needs to be made for querying the values of designer-controlled variables as part of each program run. The top-down nature of the system means that designer-controlled variables would otherwise each only be assessed once after starting with the scratch case study (case study 1 with all NPRD=0) because, at the first accessing, the presence of data flags of the variable and its dependents would be set to valid and TABEX3 would have no cause to descend to lower levels of the data network. Four ways of giving the designer access to designer-controlled variables are:

(a) to scan the global dependence of a variable for designer-controlled variables and allow the designer to change their values before TABEX3 requires the value of the variable

(b) to use a bottom-up method, such as setting all NPRD to 0 after a run, to force TABEX3 to access all designer-controlled variables

(c) to set up a list or lists of designer-controlled variables that can be used to identify the relevant variables before or during a program run

(d) to allow the designer to nominate variables whose values are to be changed.

Options (a) and (b) are wasteful of computing resources and in option (b) the strategy of only recalculating data where necessary is lost. Hence, only options (c) and (d) are viable. Option (c) requires careful setting up so that the querying process is suitably timed and so

only relevant data is queried; this option requires more system programming but, because the designer does not have to key in a potentially large number of names at the beginning of a program run, the time taken and the number of potential errors is reduced. Option (c), in which the system presents the designer with variables for possible alteration, was selected for the present project because of its lesser demands on computer processing time and overall faster pre-run set-up time. Option (d), in which the designer presents the system with the names of variables for alteration, has the advantage that the designer only reviews the data he wants to. By allowing a revocable suppression of the querying of the values of rarely altered variables or constants, the advantage of option (d) was incorporated into option (c).

The simplest way to create a data review facility is to form a list of all designer-controlled variable storage addresses and, at the beginning of each run, step through the list giving the designer the option of retaining the present value or entering a new value for each variable. However, with all but the smallest programs this would result in review of data irrelevant to the particular program, which would prolong the pre-run set-up time and possibly cause confusion. Therefore, the single list must be replaced with multiple lists which must be accessed before data which is dependent on any variable on the list is used by TABEX3. Reviewing data before the run starts is safe but, as TABEX3 only processes necessary data and as the sequence of operations will thus vary from one run to the next, the number of designer-controlled variables that must be reviewed will also vary. The lists of designer-controlled variables must therefore be distributed throughout the program and data reviewed at intervals during the program

run, according to the path taken by TABEX3. The review process must be triggered when TABEX3 encounters a variable or decision table dependent on the listed variables. If the lists were to be associated with variables dependent on a listed element they would have to be linked with the highest level variables that TABEX3 would have to recalculate if the value was changed by the designer. If the lists were placed at an element lower in the data network, a catch-22 situation would result: the review process would not be carried out because higher level data were defined, but this data would not be invalidated unless the value of a designer-controlled variable was altered when a list of variables was reviewed. Because the highest level variables will be expressions in the condition or action stubs of decision tables, it was decided to associate the lists with decision tables rather than with variables.

The global dependence of a single element is likely to contain expressions from more than one decision table, therefore that element may appear on more than one designer-controlled variable list. To prevent the value of a variable being reviewed more than once during a single program run, TABEX3 sets the presence of data flag to valid after the first query (whether or not the value of the variable has been altered) and only reviews designer-controlled variables with presence of data flags set to void.

Designer-controlled variable lists are difficult to build during program generation because it is not possible to trace the global dependency of a data item when the item is first encountered by the system during program input as a result of the incomplete nature of the data network at this stage. Additionally, subsequent modifications to

decision tables or data relationships would require further processing to identify areas affected by changes and then make the necessary corrections. Therefore, designer-controlled variable lists are built during run-time.

The run-time approach is based on a zero value for each presence of data flag for a new data network; this forces TABEX3 to traverse the lower levels of the network during decision table execution. By noting the number of the most recent decision table being executed, any designer-controlled variables can immediately be added to the list associated with that table. Similarly, when an existing program is modified, any new designer-controlled variables will be encountered by TABEX3 and must be processed before dependent data can be evaluated.

A minor weakness of the run-time strategy is that if the modify option is used to add a new decision table to a program and if the new table has a designer-controlled variable in the global ingredience of its condition expression which is common to an existing table, then if the existing table is normally executed first, the common variable will not be put onto the list associated with the new table as it will be queried before execution of the new table begins. This is not a problem in the normal course of events because all information is available to both tables when required. However, if the designer elected to execute the new table directly using a case study in which data depending on the designer-controlled variable is already defined, the variable will not be queried. This is easily overcome by re-running the table using the scratch case study (case study one with all NPRD=0) or nominating the unlisted variable at the 'query/don't query' run-time option and then

re-running the table. A message to this effect is incorporated into the system programs.

When a table is executed, the presence of immaterial entries (-) in the condition entry may mean that not all of the individual condition expressions need to be evaluated before the appropriate rule is found. Because all designer-controlled variables on the list associated with a table are reviewed before the table is executed, the values of some data will be needlessly queried. This could be overcome by associating a designer-controlled variable list with each table row (conditions and actions) rather than using a single list attached to the table itself.

4. QUERY / DON'T QUERY OPTION

As a design progresses, the designer will be adjusting the values of a progressively smaller number of parameters. If the problem contains a large number of variables it is convenient to review only the values of relevant variables. This is achieved by using a query/don't query option which suppresses the review of nominated elements by setting their presence of data flags to valid (NPRD=1). The TABEX3 designer-controlled variable query process will then recognise the value of the variable as valid and use it without further evaluation.

As the system requires the use of named constants instead of actual numbers, the query/don't query option is particularly useful as it can be used to prevent the repeated querying of an unchanging value.

If the designer wishes to reactivate the querying of a particular

designer-controlled variable, the status of the variable is reset to void (NPRD=0) and WARN3 is used to clear all data in the variable's global dependency.

This method of control affects only the presence of data flags, so a designer-controlled variable which is queried in one case study can be suppressed in another. Thus, if there are two or more major solution options which concentrate on different groups of data elements, separate case studies can be set up to manipulate only the data relevant to each option.

5. OVERRIDING THE VALUE OF A VARIABLE

To help solve a design problem, the designer may wish to force a variable to take on a particular value rather than the value which would be assigned by calculation or decision table. This override option could be used to test the logic or sensitivity of a particular program or part of a program, to isolate the effect of a particular parameter or decision, or to move more quickly towards a solution through enforcing a coarse model of the design problem, thus avoiding having to refine the full designer-controlled variable set until close to a possible solution.

Override is achieved by entering the new variable value and then setting the presence of data flag (NPRD) to '2'. This flag value avoids the invalidation of the overriden value by WARN3 clearing data having NPRD = 1, and allows the value to be used by TABEX3 which will recalculate the value of a variable if its NPRD is non-zero.

The address of an overridden variable is added to the reset list used for designer-controlled variables so that its status is reset to void at the conclusion of the design iteration. This is a safety measure to avoid an overriden variable being overlooked when a seemingly satisfactory solution is reached. In addition, a reminder is given to the designer when the override facility has been used during a design iteration.

The override feature adds to the versatility offered by the 'Program Generator For Designers' suite of programs and is one which is not routinely available to users of programs written using conventional programming methods.

6. QUERYING THE CALCULATED VALUE OF A VARIABLE

A decision table action option is to display the current value of a nominated variable. However, after completing an application program run the designer may require additional information to assist in preparation of data for the next run or may wish to check the effects of program parameters on a particular variable. This information is provided by requesting the designer to enter the variable name and then retrieving the value from the storage address in CASE given by the INDEX algorithm. A 'variable currently undefined' message is given if the status indicator at the location is set to void.

CHAPTER VIII

IMPLEMENTATION

Computer programs to implement the Program Generator were written in FORTRAN and were developed largely on an IBM PC-XT with the balance of the work being done on a Micro VAX. The Program Generator does not use any computer system-dependent commands. Program relationships within the Program Generator suite of programs are shown in Appendix A and program listings can be found in Appendix B. Each program incorporates documentation to explain its function, method and relationship to other programs and data files. Program and subroutine names reflect the algorithm names from Chapters VI and VII.

The user of the system (the designer) interacts with the system programs, firstly, to generate an application program using decision tables to represent the design logic processes and ,secondly, to run the decision tables to solve the design problem, modifying the tables and other data as necessary to refine or correct the problem model. The system is completely menu-driven and, through the menus and other system prompts, guides the designer through the generation and use of the application program. The manner of system operation is represented in the system menus listed in figures 8.1 to 8.7 and can be further seen in the worked example of Appendix C.

During program generation and run-time the system uses a combination of array (memory) and file (disk) storage to hold the decision tables and other data.

Application program output is written to the sequential data file OUTPUT.dat which cannot be printed from within the compiled FORTRAN programs and must be spooled by the designer from the operating system of the computer. Output is sourced from decision table actions which, in the case of action types which show the values of variables or give messages, are written to both file and screen. The values of all designer-controlled variables can be recorded on file so that the designer can retain a permanent record of both program input and output.

MAIN MENU

Do you wish to -

1 Run an existing program or view data, tables etc.

2 Modify an exisiting program or data

3 Create a new program

4 Quit via save option

:: enter number of choice and press <RETURN>

Figure 8.1 Main Menu (from program DESIGN)

Notes :

option 1 - refer to figure 8.2 for sub-menu in subroutine DESRUN

option 2 - refer to figure 8.4 for sub-menu in subroutine MODIFY

option 3 - no sub-menu is used

RUN MENU

Do you wish to -

1 select and run a program

2 re-run the last program

3 select a case study

4 save the current case study

5 change variable status to query/don't query each run

6 preset (override) the value of a variable normally calculated by the program (Resets after next run)

7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry

0 return to the main menu

:: enter number of choice -

Figure 8.2 Run Menu (from subroutine DESRUN)

Notes :

option 1 - lists all decision tables and requests selection

option 2 - avoids having to reselect the decision table each run

option 3 - lists all case studies and their current status

option 4 - selection of storage case study for data in work area

option 5 - suppresses/reactivates querying of a variable each run

option 6 - use a designer selected value for a variable

option 7 - refer to figure 8.3 for View Menu in subroutine DESRUN

VIEW MENU

Do you wish to -

1 view a decision table

2 view the current value of a variable

3 view all equations associated with a variable

4 view a look-up table entry

0 return to previous menu

:: enter number of choice -

Figure 8.3 View Menu (from subroutine DESRUN)

Notes :

option 1 - uses subroutine SHOW

option 3 - lists all the equations used to calculate the variable

option 0 - returns to the RUN MENU

MASTER MODIFICATION MENU

Do you wish to alter a -

1 decision table

2 variable (units or equation)

3 look-up table

4 quit modification, return to main menu

:: enter number of choice -

Figure 8.4 Modification Menu (from subroutine MODIFY)

Notes :

option 1 - refer to figure 8.5 for sub-menu

option 2 - refer to figure 8.6 for sub-menu

option 3 - refer to figure 8.7 for sub-menu

(all sub-menus are in subroutine MODIFY)

```
 Which aspect of the decision table is to be modified?
1  entire table (delete)
2  condition     (delete, append, re-enter)
3  action        (delete, append, re-enter)
4  rule          (delete, append, re-enter)
5  return to master modification menu
6  quit modification, return to main menu
:: enter number of choice -
```

Figure 8.5 Decision Table Modification (from subroutine MODIFY)

Notes :

option 1 - deletes the nominated table after querying user

options 2,3,4 - use a sub-menu to select from the three options

Which aspect of the variable is to be modified?

1 units

2 an equation

3 return to master modification menu

4 quit modification, return to main menu

:: enter number of choice -

Figure 8.6 Variable Modification (from subroutine MODIFY)

Notes :

option 2 - requires re-entry of the nominated equation

Do you wish to -

1 re-enter a single table entry

2 re-enter an entire look-up table

3 delete an entire look-up table

4 return to master modification menu

5 quit modification, return to main menu

:: enter number of choice -

Figure 8.7 Look-up Table Modification (from subroutine MODIFY)

Notes :

option 3 - deletes after querying

CHAPTER IX

DISCUSSION

1. DECISION TABLES AND THE PROGRAM GENERATOR AS DESIGN TOOLS

Decision tables have been used as programming medium in the past but the extent to which they have retained their form has varied.

The approach of using specialised processors such as DETAB-X (Pollock, 1971) or FORTAB has been used to convert decision tables into conventional FORTRAN or COBOL program code. This recognises the potential of decision tables for documenting and communicating design logic.

The strategy of retaining the decision table as data has been used by Goel and Fenves (1971) but, although this particular work incorporated a general purpose processor to scan the entry parts of decision tables, FORTRAN coded statements specific to the application had to be provided to express the condition and action stubs, and to relate the variable names to their array storage locations.

The Program Generator For Designers incorporates a completely general purpose decision table interpreter and, as no recourse is made to the use of FORTRAN or other program code to express any part of the problem, the user need not possess any computer programming skills. The Program Generator removes the need for manual preprocessing of the decision tables or other data, thus, the user does not even have to be

aware of the internal workings of the system. The only requirement made of the user is that he must have sufficient knowledge of decision tables to enable him to express the design problem in decision table form. Clearly, becoming familiar with decision tables requires considerably less effort than does gaining proficiency in programming in a language such as FORTRAN.

Having mastery of a computer programming language does not necessarily imply that good programs will result, or that programming the application will be straightfoward. The advantages of using the Program Generator For Designers to create an application program come not only from the benefits of using decision tables but also from its use of a top-down programming method. The user of a traditional programming language must take time to order his data, inputs and decisions before even commencing the coding process. The Program Generator For Designers, however, requires the designer to prepare his decision points only so far as they affect a particular variable course of action, or decision (such as the ultimate decision on whether the design is satisfactory or not). Preparation of a full program strucure is not needed because once the Program Generator is given the title of the top-most decision table by the designer, the remaining program input and generation is controlled entirely by the system and the designer is required only to feed information when prompted to do so. In short, the designer needs little more than a good understanding of the problem itself in order to use the Program Generator system.

Once the program is entered there is no repeated compiling and linking of program modules as syntax and other errors are progressively

removed from the program files: the program is ready to run as soon as entry (and the simultaneous generation) is completed.

Logical or mathematical faults in the generated program can be easily corrected using the menu-driven modification routines. Once the basic modifications are made the system takes over to enumerate any new decision tables or other data using the same routines as were utilised during generation of the original program. Changing or correcting a traditional computer program involves editing the program files, and then recompiling and relinking. Because modifications can easily be made to a decision table-based program the designer can choose to start with a more basic model of the problem and make a greater number of enhancements to selected parts of it than would otherwise be the case.

Run-time features which control the setting of the values of designer-controlled and ordinary variables, and the ability to run subunits of the complete program (individual or groups of tables) give the designer a flexibility not normally found in conventional computer programs.

The combination of straightforward program generation and modification features, and flexible run-time data management options gives the designer the potential to create better designs because, for the same usage of creative and computing resources as by other methods, a greater number of solution options may be investigated. The designer can concentrate on the problem in hand rather than the management of computer-aids and need not have his creativity impaired by system inflexibility.

Decision tables are well suited to expressing codes and standards (Goel & Fenves, 1971) and as such should find wide application to engineering design work where extensive constraint checking is often required.

The use of a common database can help prevent the fragmented development of application programs. Provided separate programs use common names for the same variables they can be readily linked, or used as building blocks in the development of larger systems even though they can still be run as separate units.

The documentation inherent in decision table-based application programs reduces the likelihood of a program written by one user being used out of context by another user.

The Program Generator could be usefully applied in the university learning environment. The 'black box' approach, which involves students using the output from prewritten programs, can be avoided as can the requirement for students to have adequate programming skills in a programming language such as FORTRAN. Use of the Program Generator allows knowledge of the method of solution, expressed in decision tables, to be examined along with the solution itself.

2. SYSTEM DEVELOPMENTS

The progress to date in the development of a decision table-based program generator and interpreter is encouraging and suggests that

further development to increase its versatility would be worthwhile. Some improvements, such as database management, would require specialist computer science input. Improvements such as decision table preprocessing would be beneficial to the system as it stands, while others, such as full screen data entry and editing, would not be warranted unless they were part of a computer science exercise or if major system expansion or commercial development was anticipated.

(1) Decision Table Preprocessing

Re-ordering the rows and columns of a decision table does not affect the table's logic and a planned reorganisation can reduce the time taken to process a table. A preprocessor, possibly based on the tactics discussed by Humby (1973), would be a valuable addition to the Program Generator. The preprocessor would be called from the DETAB subroutine at the completion of each decision table input or modification.

(2) Loop Detection

It is possible to set up cyclic dependences (A = B, B = A) which will go undetected until the TABEX3 or WARN3 subroutines enter an endless loop. System programs could be written to check for such relationships through checking that a variable does not figure in its own global dependence or ingredience. However, the disadvantage of such a check is that the time taken would be considerable if each variable of a large data network is to be checked, particularly if trans-table relationships are to be considered. Checking speed could be increased by improved data storage and access methods, or if the system was rewritten in a language that ran faster than the present FORTRAN version.

(3) Data Storage

When the program generator is run some data is loaded from the data files into memory arrays and variables, while other data remains on (disk) file until needed. For a system containing a large number of application programs, a high proportion of the decision tables loaded into memory might not be required during the running of a particular program, but leaving too much data on the disk would increase I/O time. The optimum balance between memory and file storage will depend on the amount of data in the system and memory availability.

Database management is one aspect of the Program Generator which would benefit from some specialised knowledge to determine the most suitable data storage and search and access methods (for the directory file in particular, but also for look-up table storage).

In the Program Generator, the requirement for upper case responses to system prompts is necessitated by the inability of the system to readily convert application program variable and look-up table names to a standard upper or lower case format. Thus, an existing name re-entered in format different to the original would be treated as a new variable. Removing this restriction would give the designer a little more freedom and remove a potential source of confusion.

With large scale applications it would be an advantage if separate files were used for each application, to reduce the amount of data in the system at any one time and to avoid unintended interactions between data items of the same name but unrelated applications. However,

access to standard data, e.g. steam tables or tables of structural steel section properties, should be maintained.

(4) Expanded Function Repertoire

The system has space for up to 35 mathematical functions and operations; 23 spaces are in current use. Adapting the system to include other functions, such as hyperbolics, requires the addition of the appropriate keyword and its assigned numerical code to the list of keywords on the KEYWRD.dat file, the inclusion of the appropriate entries between new and existing functions in the checking matrix (which is stored on the same file) and the addition of program code to the TABEX3 subroutine to evaluate the function.

If, in the future, there is a need to enlarge the system to allow for an increased number of operators, functions or look-up tables, small changes would be required to a number of subroutines, most notably TABEX3, and the system memory variables stored at the head of the TABLE.dat file. Care should be taken to ensure that the relative priorities of functions and operators, as coded in KEYWRD.dat, are not upset. To make the addition of new functions easier, a dual coding system could be introduced: one code would be a function or operator identification number and the other would be a precedence rating. Hence, new functions could be added without needing to reallocate the existing codes.

(5) Arrays, Look-up Tables And Subscripted Variables

Look-up tables are currently read only. Expanding the system to accomodate a more generalised system of read/write arrays or, more

particularly, subscripted variables, is highly desirable and would significantly increase the Program Generator's suitability for application to more extensive design problems. The enhanced system would effectively use an array in place of the present single storage location for unsubscripted data elements. Each array dimension would correspond to a particular subscript, such as member number, loading number, joint number, and so on. Dependence and ingredience relationships would become more specialised so that links between data elements would be based on common attributes of the data elements. Hence, if the load on a particular beam in a structure is changed, data associated with other members in the structure which are affected by the changed load case are cleared. Some conceptual work has been done in this area by Wright, Boyer and Melin (1971).

(6) Solving For Unknowns

While solving a problem, the designer may wish to set the value of the left-hand side of an equation and solve for an unknown on the right-hand side. Introducing the capacity to solve for unknowns on a small scale would be comparatively straightfoward; however, if logical information is involved (across decision tables) solving would be more complex.

(7) Override Facility

The present override facility allows an override to remain in force for only one iteration. The system could be expanded to allow an override to remain active until specifically removed by the designer. A typical application of the continuing override would be in the situation

where the designer's program normally calculates member properties from base data, but the calculated data are available directly from another source such as manufacturers catalogues. An effective warning system would still have to be maintained to remind the designer of possible inconsistencies arising from isolating a part of the data network from its ingredients. Overridden variables could be treated as if they were designer-controlled and become part of the data review process during each program run.

(8) Full Screen Data Entry And Editing

The present Program Generator, because of its FORTRAN base, requires decision table edit locations to be identified prior to actual editing taking place. Decision table and data entry and modification could be greatly enhanced by introducing full screen editing. This would allow the designer use the keyboard arrow keys to move the cursor to the location of information to be modified in any of the quadrants of a displayed decision table. Editing would thus be more direct and rapid.

The full screen approach could be extended to data entry during program generation. The designer would be presented with a blank decision table to fill in which could be enlarged or contracted at will. Entry and editing of look-up table data would also be more straightforward than under the present system of single data element manipulation.

Whereas the present system requires re-entry of incorrect equations, a full screen system would enable an equation requiring correction or modification to be displayed and altered on the screen.

Developing the full screen scenario further would allow the designer to use preset keys to move around the data network and decision tables. Thus, if operating in run mode, the designer could nominate a particular data element or decision table as a starting point, and then move about the network to override the values of variables, or review the values of desinger-controlled variables. In modification mode, the designer could view and edit the equations associated with each data element, and alter any decision tables which were reached either by traversing the network or by direct selection.

(9) Data Security

Dependent on the scale of system use, the ability to lock specific resources would be an aid to protecting data from alteration by unauthorised users. Standard decision tables and other data reflecting management policy or codes and standards could then be incorporated in an application program, but could not be altered by the designer.

(10) Hard Copy Of Program Structure

The present Program Generator has full facilities to enable a designer to view data, equations, look-up table data and decision tables. This facility could be extended so that printouts of the program structure could be obtained by the designer. Further development of the printout system would allow the state of the program at that point to be viewed by highlighting the last rules applying in decision tables, the last equations used to calculate variables, and the relevant values of all variables. Hardcopy of this type would be of value when documenting a final design solution or when submitting a proposal for approval.

(11) Commercial Development

The Program Generator For Designers offers its users a flexible CAD tool without the requirements for the user to have great programming expertise. When combined with some of the proposed enhancements, this basic characteristic suits the Program Generator to a wide range of users, and hence offers the possibility for development of a commercially viable package. The range of PC users and uses suggests that development of a PC-based package would be most appropriate. Should commercial development be undertaken, attention should be paid to the major development proposals of re-evaluating data storage, accomodating the use of subscripted variables, and full screen data entry and editing.

CHAPTER X

CONCLUSION

This project has developed a CAD package to enable a designer without conventional computer programming skills to generate and run an application program which, unlike traditional methods, is based on decision tables. The use of decision tables allows better documentation, communicatation and revision of the design logic than do traditional methods.

The Program Generator For Designers is an interactive, menu-driven system which is completely independent of any of its applications. The designer is given a great deal of flexibility, both in expressing and modifying the design logic, and in manipulating the values of the design parameters.

The current project could form the basis for the development of a more extensive system with the potential for commercial application.

ACKNOWLEDGEMENTS

This research project was carried out under the supervision of Professor H. McCallion whose inspiration and constructive guidance is gratefully acknowledged.

Access to personal computing facilities, on which most of the work took place, was given by the author's employer, the N.Z. Railways Corporation.

REFERENCES

(i) FENVES, S.J. (1966) Tabular design logic for structural design. Journal of the Structural Division, Proceedings of the American Society of Civil Engineers, 92: 473-490.

(ii) FENVES, S.J. (1972) Representation of the computer-aided design process by a network of decision tables. Proceedings of the International Symposium on Computer-Aided Design 1: E1.31-E1.45.

(iii) GOEL, S.K. and FENVES, S.J. (1971) Computer-aided processing of design specifications. Journal of the Structural Division, Proceedings of the American Society of Civil Engineers, 97: 463-479.

(iv) HUMBY, E. (1973) Programs from decision tables. MacDonald/American Elsevier.

(v) McDANIEL, H. (1968) An introduction to decision logic tables. New York, John Wiley & Sons.

(vi) POLLOCK, S.L. (1971) Decision tables: theory and practice. New York, Wiley-Inter Science.

(vii) ROARK, R.J. and YOUNG, W.C. (1975) Formulas for stress and strain. 5th ed. Auckland, McGraw-Hill.

(viii) SMITH, G.E. (1986) Are CAD users missing the 'big picture'? Machine Design, 58 (No.12): 204.

(ix) WRIGHT, R.N., BOYER, L.T. and MELIN, J.W. (1971) Constraint processing in design. Journal of the Structural Division, Proceedings of the American Society of Civil Engineers, 97: 481-494.

APPENDICES

APPENDIX

APPENDIX A

PROGRAM STRUCTURE

This appendix displays the program structure which makes up the Program Generator For Designers. The names of any subroutines called by a program are shown to the right of the title of the program.

DESIGN - DESPGM
- MODIFY
- DESRUN

DESPGM - DETAB
- ENCODE
- DCVMSG
- LKLOAD
- INPUT

DESRUN - GETLK
- GETTAB
- GETVAR
- GETXYZ
- LKPOSN
- SHOW
- TABEX3
- WARN3

ENCODE - DEPLST
- LOOK
- RPN
- XINDEX

GETVAR - XINDEX

XINDEX - INSERT

RPN - EQN

DETAB - ENCODE

- INPUTS
- XINDEX

MODIFY - DESPGM

- SHOW
- DETAB
- GETVAR
- INPUT
- ENCODE
- UNDEPN
- GETLK
- GETXYZ
- LKPOSN
- LKLOAD

APPENDIX B

PROGRAM LISTINGS

This appendix contains listings of the FORTRAN programs written to implement the Program Generator. The parent program, DESIGN, is given first, followed by the subroutine listings in alphabetical order of subroutine name.

```
      PROGRAM DESIGN
*************************************************************************
* Parent program for program suite "A Program Generator For Designers"
*
* PURPOSE : performs the systems I/O work and runs the main menu.
* ROUTINES CALLED :
*     DESRUN - runs an existing program
*     DESPGM - controls entry of a new program
*     MODIFY - additions and alterations to existing programs
* COMMON ARRAYS :
*     NCOND, NACT, NRULE - decision table dimensions.
*     NTYPE  - type of each decision table action.
*     NSTORE - condition & action stub pointers indicating
*              storage location of condition or action.
*     METHOD - method of calculation of a variable ie. equation to
*              be used.
*     NDCV   - list of Designer Controlled Variables for each
*              table.
*     TABLE  - condition and action entries.
*     TITLE  - decision table titles.
*     MESAGE - decision table messages
*     ISIZE  - length of keywords
*     ICODE  - keyword code numbers
*     ICHEC  - checking array used during encoding
*     STRING - input equation
*     TEST   - keywords
*     LKDIM  - no. of dimensions to each look-up table
*     LKSIZE - size of look-up table dimensions
*     LKPTR  - start positions of each tables data in RLKDAT
*     LKTOT  - number of entries in look-up tables
*     RLKDAT - look-up table contents
*     LKNEXT - next vacant slot in RLKDAT
*     LKAXIS - look-up table dimension titles
*     LKNAME - look-up table names
*     CASE   - case study titles
*     NPRD   - presence of data flags
*     RCASE  - case study data
*     LASTRL - last rule applying for each decision table in each case
*     LASTEQ - last eqn used to calculate each variable
*     NRESET - list of variable addresses whose NPRD are to be reset
*     NRPNEQ - Reverse Polish form of equation
* COMMON VARIABLES :
*     NENTRY - total no. variables in system
*     NPLACE - variable no. where unprocessed variables begin
*     NDETAB - total no. decision tables in system
*     NXTAB  - next table to be processed
*     NMESGE - number of messages in system
*     NUMEQN - number of equations in system
*     NKEYWD - no. of keywords in system
*     NLKUP  - no. of look-up tables in system
*     LKTODO - flag indicating whether look-up tables await processing
* DATA FILES REFERENCED :
*     CASE.DAT   - variable values stored according to case study.
*     KEYWRD.DAT - reference file for scanning of input equations.
```

```
*     OUTPUT.DAT - results output from DESRUN.
*     TRACE.DAT  - records messages tracing progress of TABEX3 & WARN3
*     FIXED.DAT  - eqn, and decision table numbers and dependence lists,
*                  names and units associated with each variable.
*     DIRECT.DAT - directory of variable storage locations.
*     ARRAY.DAT  - look-up tables.
*     CALC.DAT   - equations for variable evaluation - original &
*                  Reverse Polish forms.
*     TABLE.DAT  - systems data, designer controlled variable lists,
*                  decision table entries, decision table messages.
* VERSION /DATE : 1.00 /  1- 6-88
* PROGRAMMER    : A.A.Hunt
* REMARKS :
*     PREFIXES   - K  DO loop counter
*     Decision Tables - condition and action stubs store addresses only.
*     Implicit naming of variables applies throughout.
*
************************************************************************
      CHARACTER*30 TITLE(100),CASE(6),MESAGE(100)
      CHARACTER*20 LKAXIS(15,5),LKNAME(15)
      CHARACTER*1 UPPER
      CHARACTER TEST(35,5),NAME(20),UNITS*10,STRING(80),TABLE,REPLY*1
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK2/ISIZE(35),ICODE(35),ICHEC(35,35),NKEYWD
      COMMON/BLOK3/STRING,TEST
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(1000),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
      COMMON/BLOK8/NDCV(100,30)
C     COMMON/BLOK9/NRPNEQ(50)
C
      OPEN(8,FILE='CASE.DAT',STATUS='OLD')
      OPEN(9,FILE='KEYWRD.DAT',STATUS='OLD')
      REWIND(9)
      OPEN(10,FILE='TRACE.DAT',STATUS='UNKNOWN')
      CLOSE(10,STATUS='DELETE')
      OPEN(10,FILE='TRACE.DAT',STATUS='NEW')
      OPEN(11,FILE='OUTPUT.DAT',STATUS='UNKNOWN')
C      OPEN(11,FILE='OUTPUT.DAT')                              PC line !!
      CLOSE(11,STATUS='DELETE')
      OPEN(11,FILE='OUTPUT.DAT',STATUS='NEW')
      OPEN(12,FILE='FIXED.DAT',ACCESS='DIRECT',FORM='FORMATTED',
     +     RECL=441,STATUS='OLD')
      OPEN(13,FILE='DIRECT.DAT',ACCESS='DIRECT',FORM='FORMATTED',
     +     RECL=26,STATUS='OLD')
      OPEN(14,FILE='ARRAY.DAT',RECORDTYPE='VARIABLE',
     +     RECL=6000,STATUS='OLD')
```

```
C      OPEN(14,FILE='ARRAY.DAT',STATUS='OLD')                PC line !!
       REWIND(14)
       OPEN(15,FILE='CALC.DAT',ACCESS='DIRECT',FORM='FORMATTED',
      +     RECL=402,STATUS='OLD')
       OPEN(16,FILE='TABLE.DAT',STATUS='OLD')
       REWIND(16)
C
C      * put up title *
       WRITE(*,19)
C      * read test keywords in from file *
       READ(9,20)NKEYWD
       DO 46 K46=1,NKEYWD
          READ(9,21)ISIZE(K46),(TEST(K46,J),J=1,5),ICODE(K46)
   46  CONTINUE
C      * read checking matrix in from file *
       READ(9,20)NCHEC
       DO 47 K47=1,NCHEC
          READ(9,22)(ICHEC(K47,J),J=1,NCHEC)
   47  CONTINUE
C
C      * read in systems data *
       READ(16,10)NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
C
C      * read in case study status *
       DO 41 KCASE=1,6
          READ(16,17)CASE(KCASE)
   41  CONTINUE
C      * read in case study data *
       DO 415 KREC=51,NENTRY
          READ(8,28)(NPRD(JCASE,KREC),JCASE=2,6),
     +       (RCASE(JCASE,KREC),JCASE=2,6)
  415  CONTINUE
C      * read in last rule array *
       DO 50 KTAB=1,NDETAB
          READ(16,27)(LASTRL(KCASE,KTAB),KCASE=2,6)
   50  CONTINUE
C
C      * read in last eqn array *
       DO 52 KEQN=1,NENTRY
          READ(16,252)(LASTEQ(KCASE,KEQN),KCASE=2,6)
   52  CONTINUE
C

C      * read in messages *
       DO 48 KMESGE=1,NMESGE
          READ(16,17)MESAGE(KMESGE)
   48  CONTINUE
C
C      * read in decision tables and associated information *
       DO 44 K44=1,NDETAB
          READ(16,15)KTAB,NCOND(KTAB),NACT(KTAB),NRULE(KTAB)
          READ(16,17)TITLE(KTAB)
C         * read in condition stub and entries *
```

```
            DO 451 KROW=1,NCOND(KTAB)
               READ(16,161)NSTORE(KTAB,KROW),
     +                     (TABLE(KTAB,KROW,J),J=1,NRULE(KTAB))
  451       CONTINUE
C           * read in action stub and entries *
            DO 45 KACT=1,NACT(KTAB)
               KROW=NCOND(KTAB)+KACT
               READ(16,16)NTYPE(KTAB,KACT),NSTORE(KTAB,KROW),
     +                    METHOD(KTAB,KACT),
     +                    (TABLE(KTAB,KROW,J),J=1,NRULE(KTAB))
   45       CONTINUE
C           * read in DCV list *
            READ(16,18)NDCV(KTAB,1),(NDCV(KTAB,J),J=2,NDCV(KTAB,1)+1)
   44    CONTINUE
C
C        * read in look-up table data *
         LKNEXT=1
         DO 412 KLOOK=1,NLKUP
            READ(14,24)LKNO,LKDIM(LKNO)
            IF(LKDIM(LKNO).GT.0)THEN
C              * table is active - read in remaining data *
               READ(14,251)LKNAME(LKNO),LKTOT(LKNO)
               READ(14,252)(LKSIZE(LKNO,J),J=1,LKDIM(LKNO))
               READ(14,253)(LKAXIS(LKNO,J),J=1,LKDIM(LKNO))
C              * read in data for look-up table *
               LKPTR(LKNO)=LKNEXT
               NHOLD=LKPTR(LKNO)+LKTOT(LKNO)-1
               READ(14,26)(RLKDAT(J),J=LKPTR(LKNO),NHOLD)
               LKNEXT=NHOLD+1
            ELSE
               LKPTR(LKNO)=0
            ENDIF
  412    CONTINUE
C
C        * reading in of decision tables, data etc.ends *
C
C        * check input is upper case *
   91    WRITE(*,29)
         READ(*,17)UPPER
         IF(UPPER.NE.'Y')GOTO 91
C        * select option to run, create, modify etc. *
   70    WRITE(*,11)
         READ(*,12,ERR=70)NANSWR
         GOTO(71,72,73,74)NANSWR
         GOTO 70
C
C        * run an existing program *
   71    CALL DESRUN
         GOTO 70
C
C        * edit *
   72    CALL MODIFY
         GOTO 70
```

```
C
C     * enter a new program *
   73 NDETAB=NDETAB+1
      WRITE(*,13)
      READ(*,17)TITLE(NDETAB)
      CALL DESPGM
      GOTO 70
C
C     * quit via save option *

C     * write all data back to files *
   74 REWIND(16)
C     * write systems data back to file *
      WRITE(16,10)NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,
     +            NLKUP,LKTODO
C     * write back case study status *
      DO 411 KCASE=1,6
         WRITE(16,17)CASE(KCASE)
  411 CONTINUE
C     * write back case study data *
      REWIND(8)
      DO 416 KREC=51,NENTRY
         WRITE(8,28)(NPRD(JCASE,KREC),JCASE=2,6),
     +      (RCASE(JCASE,KREC),JCASE=2,6)
  416 CONTINUE
C     * write last rule array back to file *
      DO 51 KTAB=1,NDETAB
           WRITE(16,27)(LASTRL(KCASE,KTAB),KCASE=2,6)
   51 CONTINUE
C     * write last eqn array back to file *
      DO 521 KEQN=1,NENTRY
         WRITE(16,252)(LASTEQ(KCASE,KEQN),KCASE=2,6)
  521 CONTINUE
C
C     * write back messages *
      DO 481 KMESGE=1,NMESGE
         WRITE(16,17)MESAGE(KMESGE)
  481 CONTINUE
C     * write back decision tables *
      DO 42 KTAB=1,NDETAB
         WRITE(16,15)KTAB,NCOND(KTAB),NACT(KTAB),NRULE(KTAB)
         WRITE(16,17)TITLE(KTAB)
         DO 431 KROW=1,NCOND(KTAB)
            WRITE(16,161)NSTORE(KTAB,KROW),
     +                   (TABLE(KTAB,KROW,J),J=1,NRULE(KTAB))
  431    CONTINUE
         DO 43 KACT=1,NACT(KTAB)
            KROW=NCOND(KTAB)+KACT
            WRITE(16,16)NTYPE(KTAB,KACT),NSTORE(KTAB,KROW),
     +                  METHOD(KTAB,KACT),
     +                  (TABLE(KTAB,KROW,J),J=1,NRULE(KTAB))
   43    CONTINUE
C        * write DCV list back to file *
```

```
         WRITE(16,18)(NDCV(KTAB,J),J=1,NDCV(KTAB,1)+1)
   42 CONTINUE
C     * write look-up table data back to file *
      REWIND(14)
      DO 414 LKNO=1,NLKUP
         WRITE(14,24)LKNO,LKDIM(LKNO)
         IF(LKDIM(LKNO).GT.0)THEN
C           * table is active - write back remaining data *
            WRITE(14,251)LKNAME(LKNO),LKTOT(LKNO)
            WRITE(14,252)(LKSIZE(LKNO,J),J=1,LKDIM(LKNO))
            WRITE(14,253)(LKAXIS(LKNO,J),J=1,LKDIM(LKNO))
C           * write back data for this table *
            NHOLD=LKPTR(LKNO)+LKTOT(LKNO)-1
            WRITE(14,26)(RLKDAT(J),J=LKPTR(LKNO),NHOLD)
         ENDIF
  414 CONTINUE
C     * writing back of information ends *
C
C     * tell how to obtain printed output *
      WRITE(*,30)
C
      CLOSE(8)
      CLOSE(9)
      CLOSE(11)
      CLOSE(10)
      CLOSE(12)
      CLOSE(13)
      CLOSE(14)
      CLOSE(15)
      CLOSE(16)
      STOP
C
   10 FORMAT(8I4)
   11 FORMAT(/1X/1X/'                          MAIN MENU'
     + /1X/' ------------------------------------------------'
     + /1X/' Do you wish to :-'/
     + /1X /' 1  Run an existing program or view data, tables etc'
     + /' 2  Modify an existing program or data'
     + /' 3  Create a new program'
     + /' 4  Quit'
     +/1X/' :: enter number of choice and press <RETURN> '$)
   12 FORMAT(I1)
   13 FORMAT(/1X/1X/' :: enter program title (up to 30 characters) -')
   15 FORMAT(I3,3I2)
   16 FORMAT(I1,I3,I3,50A1)
  161 FORMAT(1X,I3,3X,50A1)
   17 FORMAT(A)
   18 FORMAT(30I4)
   19 FORMAT(/22X,33('-'),/1X,/22X,'A PROGRAM GENERATOR FOR DESIGNERS'
     +       /1X/30X,'Version 1.0  1988'
     +       /1X/1X/22x,'Mechanical Engineering Department'
     +       /1X/28X,'School Of Engineering'
     +       /26X,'University Of Canterbury'
     +       /1X/22X,33('-'))
```

```
 20 FORMAT(I2)
 21 FORMAT(I1,1X,5A1,1X,I2)
 22 FORMAT(33I1)
 24 FORMAT(I2,1X,I2)
251 FORMAT(A20,I3)
252 FORMAT(5I3)
253 FORMAT(5A20)
 26 FORMAT(1000E12.2E2)
 27 FORMAT(5I2)
 28 FORMAT(I1,4(1X,I1),5(1X,E12.2E2))
 29 FORMAT(/1X/' Ensure shift lock is on '
   +         '(capital letters are required)!!'
   +        /' :: are you ready to continue? (Y) '$)
 30 FORMAT(/1X/' To obtain your printout of output generated this'
   +        /' session type PRINT OUTPUT.DAT when prompt appears.'/1X
   +/' The output file will be DELETED on commencing the next run')
    END
```

```
      SUBROUTINE DCVMSG
C*************************************************************************
C   PURPOSE : To give a message re getting designer-controlled
C             variables onto lists.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      WRITE(*,101)
      RETURN
C
  101 FORMAT(/1X//' INFORMATION :- variables whose values are entered'
     + ' directly by the user'
     +/' are automatically associated with any decision table which '
     +/' directly or indirectly uses them and, unless it is specified '
     +/' otherwise, a value will be requested for the variable each '
     +/' time the table is used. If the value of a relevant variable is'
     +/' not being queried, either: use the scratch case study (case 1)'
     +/' and execute the table, or use the query-don''t query option '
     +/' on the RUN MENU and then execute the table.'/1X/1X/1X)
      END
```

```
      SUBROUTINE DEPLST(LOCATE,NEQN)
C*************************************************************************
C   PURPOSE :   maintenance of dependence lists.
C   ARGUMENTS :
C               LOCATE  address of the eqn LHS which is to be added to
C                       the dependence list of each ingredient listed
C                       in the RHS.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C               NRPNEQ holds Rev. Polish form of eqn belonging to LOCATE
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :   LOCATE is also added to dependence lists of look-up
C               table arguments.
C               A variable may apear on a dependence list >1 time due
C               to different source eqns.
C*************************************************************************
      CHARACTER HOLD*41
      INTEGER NDEPN(50),NSORCE(50)
      COMMON/BLOK9/NRPNEQ(50)
      DO 42 KRPN=2,(NRPNEQ(1)+1)
         IF(NRPNEQ(KRPN).GT.50)THEN
C           * NRPNEQ(KRPN) is an address, read in dependence list *
            NREC=NRPNEQ(KRPN)
*            READ(12,10,REC=NREC)HOLD,NDEPN(1),
*     +                                    (NDEPN(J),J=2,(NDEPN(1)+1))
            READ(12,10,REC=NREC)HOLD,(NDEPN(J),J=1,50),
     +         (NSORCE(K),K=1,50)
C           * check whether LOCATE is already on dependence list *
            KEND=NDEPN(1)+1
            DO 41 K41=2,KEND
               IF(NDEPN(K41).EQ.LOCATE.AND.NSORCE(K41).EQ.NEQN)GOTO 42
   41       CONTINUE
C           * add new dependent to list *
            NDEPN(1)=NDEPN(1)+1
            NDEPN(NDEPN(1)+1)=LOCATE
            NSORCE(1)=NSORCE(1)+1
            NSORCE(NSORCE(1)+1)=NEQN
C           * write modified dependence list back to file *
*            WRITE(12,10,REC=NREC)HOLD,(NDEPN(J),J=1,(NDEPN(1)+1))
            WRITE(12,10,REC=NREC)HOLD,(NDEPN(J),J=1,50),
     +         (NSORCE(K),K=1,50)

         END IF
   42 CONTINUE
      RETURN
C
   10 FORMAT(A41,50I4,50I4)
      END
```

```
      SUBROUTINE DESPGM
C*********************************************************************
C  PURPOSE: controls entry of new program information.
C  ROUTINES CALLED : DETAB
C                    ENCODE
C                    DCVMSG
C                    INPUT
C                    LKLOAD
C  ARGUEMENTS : setup data is received via common blocks.
C  COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C  PROGRAMMER :  A.A.Hunt
C  VERSION / DATE :  1.00 /  1- 6-88
C  REMARKS : DESPGM is also called from MODIFY to process data
C            arising from program modifications.
C*********************************************************************
      CHARACTER NAME*20,UNITS*10
      CHARACTER*20 LKAXIS(15,5),LKNAME(15)
      CHARACTER*30 TITLE(100),CASE(6)
      INTEGER NDEPN(50),NSORCE(50)
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
      COMMON/BLOK8/NDCV(100,30)
C
      MIDARG=0
C     * check for unprocessed variables *
C     * (don't use a loop here as NENTRY may change before label 74) *
   70 IF(NPLACE.LE.NENTRY)THEN
C        * variables to be processed *
         READ(12,105,REC=NPLACE)NEQN,NTABD,UNITS,NAME,
     +      (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
         IF(NTABD.EQ.0.AND.NEQN.EQ.0.)THEN
C           * variable as yet undefined *
C           * NDEPN(1) = 0 implies condition stub - don't request units *
            IF(NDEPN(1).NE.0)THEN
C              * request units for name *
               WRITE(*,107)NAME
               WRITE(17,107)NAME
               READ(*,108)UNITS
C              * ask how value of variable is to be assigned *
   91          WRITE(*,109)NAME
               WRITE(17,109)NAME
               READ(*,110)NANSWR
               GOTO(71,72,73)NANSWR
               GOTO 91
            ELSE
C              * stub may be a DCV *
  911          WRITE(*,1091)NAME
               WRITE(17,1091)NAME
               READ(*,110)NANSWR
```

```
               GOTO(71,911,73)NANSWR
               GOTO 911
            ENDIF
C
C           * value to be assigned by decision table *
   71       NDETAB=NDETAB+1
            NEQN=0
            NTABD=NDETAB
            WRITE(*,111)
            WRITE(17,111)
            READ(*,108)TITLE(NDETAB)
C           * give info. re designer-controlled variables *
            CALL DCVMSG
            GOTO 74
C
C           * value to be assigned by equation *
   72       NPREVS=0
            NUMEQN=NUMEQN+1
  901       CALL INPUT(NAME)
            LENGTH=70
            CALL ENCODE(NPREVS,NUMEQN,MIDARG,LENGTH,NPLACE)
C           * check for errors during equation processing *
            IF(MIDARG.EQ.0)GOTO 901
            NTABD=0
            NEQN=NUMEQN
            GOTO 74
C
C           * value is entered by designer (variable is a DCV) *
C           * Designer will be queried (by DESRUN) whether DCV is to
C           * be set to a particular value or queried before each run *
   73       NEQN=0
            NTABD=0
C
   74       WRITE(12,105,REC=NPLACE)NEQN,NTABD,UNITS,NAME,
     +          (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
         END IF
         NPLACE=NPLACE+1
         GOTO 70
      END IF
C
C     * check for unprocessed decision tables *
      IF(NXTAB.LE.NDETAB)THEN
C        * decision tables to be processed *
         NTAB=NXTAB
C        * initialise last rule applying array *
         DO 48 KCASE=2,6
            LASTRL(KCASE,NTAB)=0
   48    CONTINUE
C        * initialise dcv list *
         NDCV(NTAB,1)=0
C
         NEDIT=0
         NWHICH=0
         CALL DETAB(NTAB,NEDIT,NWHICH)
```

```
         NXTAB=NXTAB+1
C        * have further variables arisen from table input ? *
         GOTO 70
      END IF
C
C     check for unprocessed look-up tables *
C     * LKTODO is no. of waiting look-up tables 'to-do' *
      IF(LKTODO.GT.0)THEN
C        * tables to be loaded *
C        * find look-up tables to be processed *
         DO 42 KLKNO=1,NLKUP
            IF(LKDIM(KLKNO).GT.0.AND.LKPTR(KLKNO).EQ.0)THEN
C              * look-up table is active & has not had data loaded *
               CALL LKLOAD(KLKNO)
            ENDIF
   42    CONTINUE
         LKTODO=0
         GOTO 70
      ENDIF
C
      RETURN
C
  101 FORMAT(I1,1X,5A1,1X,I2)
  102 FORMAT(I4)
  103 FORMAT(33I1)
  105 FORMAT(I4,1X,I4,1X,A10,1X,A20,50I4,50I4)
  106 FORMAT(A10,1X,A)
  107 FORMAT(//' :: enter units for ',A
     +/4X,'up to 10 characters or press <RETURN> if dimensionless -')
  108 FORMAT(A)
  109 FORMAT(/' Is ',A,' to be assigned a value by -'/1X,
     +/1X,    '1  the outcome of a decision table'
     +/1X,    '2  the result of an equation'
     +/1X,    '3  as a direct input from the designer'
     +/1X,/1X,':: enter number of choice - ')
 1091 FORMAT(/' Is ',A,' to be assigned a value by -'/1X,
     +/1X,    '1  the outcome of a decision table'
     +/1X,    '2  (option 2 not valid for a condition expression)'
     +/1X,    '3  as a direct input from the designer'
     +/1X,/1X,':: enter number of choice - ')
  110 FORMAT(I1)
  111 FORMAT(' :: enter table title (up to 30 characters) - ')
  112 FORMAT(11X,A)
      END
```

```
      SUBROUTINE DESRUN
C*********************************************************************
C   PURPOSE :  controls running of programs; pre and post-run options.
C   ROUTINES CALLED :  GETTAB
C                      TABEX3
C                      GETVAR
C                      WARN3
C                      SHOW
C                      GETLK
C                      GETXYZ
C                      LKPOSN
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :  NRESET addresses of dcv's and overrides to be
C                          reset at end of run.
C                   NCOORD used in location of look-up table data.
C   LOCAL VARIABLES :  NERROR error flag from TABEX3.
C                      NREPLY alphanumeric user responses to prompts.
C                      NANSWR numeric user responses to prompts.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  dcv = designer-controlled variable.
C*********************************************************************
      CHARACTER NREPLY*1, NAME*20, UNITS*10, EQUATN*70
      CHARACTER*30 CASE(6)
      CHARACTER*20 LKAXIS(15,5),LKNAME(15)
      INTEGER NCOORD(5)
C
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(50)
C
      NCASE=0
      NTAB=0
C
C     * give options *
   70 WRITE(*,101)
      READ(*,201)NANSWR
      IF(NANSWR.EQ.0)NANSWR=10
      GOTO(71,72,73,74,75,76,77,70,70,80)NANSWR
      GOTO 70
C
C     * select and run a program *
   71 IF(NCASE.EQ.0)THEN
C        * no case study in use *
         GOTO 73
      ENDIF
      CALL GETTAB(NTAB)
C     * check selected table valid *
   72 IF(NTAB.LE.0.OR.NTAB.GT.NDETAB)GOTO 71
      NERROR=0
      CALL TABEX3(NTAB,NCASE,NERROR)
```

```
      IF(NERROR.EQ.0)THEN
C        * reset all queried DCV's and overridden variables *
C        * give option of giving list of DCV values to output file *
C        * all overridden variables are automatically written to file *
         WRITE(*,121)
         READ(*,207)NREPLY
         IF(NREPLY.EQ.'y')NREPLY='Y'
C        * write values of DCVs to file if required *
         IF(NREPLY.EQ.'Y')THEN
            WRITE(11,124)
         ENDIF
         DO 46 KRESET=2,(NRESET(1)+1)
            READ(12,208,REC=NRESET(KRESET))UNITS,NAME
C            WRITE(*,122)NAME,RCASE(1,NRESET(KRESET)),UNITS
            IF(NREPLY.EQ.'Y')THEN
C              * write value of designer controlled variable to file *
               WRITE(11,122)NAME,RCASE(1,NRESET(KRESET)),UNITS
            ENDIF
            IF(NPRD(1,NRESET(KRESET)).EQ.2)THEN
C              * value of variable was overridden - give reminder *
               WRITE(*,123)NAME,RCASE(1,NRESET(KRESET)),UNITS
               WRITE(11,123)NAME,RCASE(1,NRESET(KRESET)),UNITS
C              * value of overridden var. may be inconsistent with
C              * values of its ingredients, warn dependents *
               NWFLAG=0
               CALL WARN3(NRESET(KRESET),NWFLAG)
            ENDIF
   46    CONTINUE
      ENDIF
      DO 47 KRESET=2,(NRESET(1)+1)
         NPRD(1,NRESET(KRESET))=0
   47 CONTINUE
C     * give option re saving output from this session *
      WRITE(*,126)
      READ(*,207)NREPLY
      IF(NREPLY.EQ.'Y')THEN
         WRITE(*,127)
         READ(*,207)NREPLY
         IF(NREPLY.EQ.'Y')THEN
            CLOSE(11,STATUS='DELETE')
            OPEN(11,FILE='OUTPUT.DAT',STATUS='NEW')
         ENDIF
      ENDIF
      GOTO 70
C
C     * select a case study *
   73 WRITE(*,105)
      DO 43 KCASE=1,6
         WRITE(*,106)KCASE,CASE(KCASE)
   43 CONTINUE
      WRITE(*,107)
      READ(*,201)NCASE
      IF(NCASE.LT.0.OR.NCASE.GT.6)GOTO 73
      IF(NCASE.NE.1)THEN
```

```
C        * load an existing case study into the work area *
         DO 42 K42=51,NENTRY
            NPRD(1,K42)=NPRD(NCASE,K42)
            RCASE(1,K42)=RCASE(NCASE,K42)
   42    CONTINUE
C        * load last rules applying for this case staudy *
         DO 421 KTAB=1,NDETAB
            LASTRL(1,KTAB)=LASTRL(NCASE,KTAB)
  421    CONTINUE
      ELSE
C        * set case study 1 presence of data flags to 0 *
         DO 422 K422=51,NENTRY
            NPRD(1,K422)=0
  422    CONTINUE
      ENDIF
      GOTO 70
C
C     * save current case study (after checking one is in use) *
   74 IF(NCASE.EQ.0)THEN
C        * no case study has been loaded *
         WRITE(*,901)
      ELSE
C        * give list of studies *
         DO 44 KCASE=2,6
            WRITE(*,106)KCASE,CASE(KCASE)
   44    CONTINUE
         WRITE(*,108)
         READ(*,201)NCASE
         IF(NCASE.LE.6.OR.NCASE.GE.2)THEN
C           * request covering label for case study NCASE *
            WRITE(*,110)NCASE
            READ(*,207)CASE(NCASE)
C           * copy data from work area *
            DO 45 K45=51,NENTRY
               NPRD(NCASE,K45)=NPRD(1,K45)
               RCASE(NCASE,K45)=RCASE(1,K45)
   45       CONTINUE
C           * file last rule applying *
            DO 451 KTAB=1,NDETAB
               LASTRL(NCASE,KTAB)=LASTRL(1,KTAB)
  451       CONTINUE
         ELSE
            WRITE(*,109)
         ENDIF
      ENDIF
      GOTO 70
C
C     * query/don't query a DCV each run *
C     * note that the status of a DCV may vary between case studies
C     * to allow specialised case studies *
C     * request variable name *
   75 CALL GETVAR(NADRES)
      IF(NADRES.GT.0)THEN
C        * valid variable name, give info. *
```

```
            CALL DCVMSG
C           * check whether a DCV *
            READ(12,206,REC=NADRES)NEQN,NTABD
            IF(NTABD.EQ.0.AND.NEQN.EQ.0)THEN
C              * variable is a DCV *
               IF(NPRD(1,NADRES).EQ.0)THEN
C                 * var is on a DCV list and queried before use by tables
C                 * whose DCV list it is on, once each run. DCV may not be
C                 * being queried by all relevant tables, even if NPRD=0.
C                 * Check if QUERY option required. If so call WARN3 *
                  WRITE(*,128)
                  READ(*,207)NREPLY
                  IF(NREPLY.EQ.'Y')THEN
                     NWFLAG=0
                     CALL WARN3(NADRES,NWFLAG)
                  ELSE
C                    * reset value *
                     WRITE(*,111)
                     READ(*,205)RCASE(1,NADRES)
                     NPRD(1,NADRES)=1
                  ENDIF
               ELSE
C                 * variable currently has a preset value. Set NPRD to 0
C                 * so value of var queried before use once each run *
C                 * (this may result in a var being added to a DCV list) *
                  NPRD(1,NADRES)=0
               ENDIF
C              * changing the value of this var may affect dependent vars *
               NWFLAG=0
               CALL WARN3(NADRES,NWFLAG)
C              * give info *
               WRITE(*,112)
            ELSE
C              * variable is not a DCV *
               WRITE(*,902)
            ENDIF
         ENDIF
         GOTO 70
C
C        * override the value of a normally calculated variable *
C        * override only lasts one run before the variable reverts to
C        * being calculated from its ingredients *
C        * give warning *
      76 WRITE(*,113)
         CALL GETVAR(NADRES)
         IF(NADRES.GT.0)THEN
C           * valid variable name *
            WRITE(*,111)
            READ(*,205)RCASE(1,NADRES)
            WRITE(*,114)
C           * set NPRD to 2 (not 1) so that WARN3 won't reset to 0 if
C           * any of the var's ingredient data changes *
            NPRD(1,NADRES)=2
C           * add var to list of vars to be reset after next run *
```

```
         NRESET(1)=NRESET(1)+1
         NRESET(NRESET(1)+1)=NADRES
C        * pause ... *
         WRITE(*,131)
         READ(*,207)NREPLY
         NWFLAG=0
         CALL WARN3(NADRES,NWFLAG)
      ENDIF
      GOTO 70
C
C     * view menu *
   77 WRITE(*,130)
      READ(*,201)NANSWR
      IF(NANSWR.EQ.0)NANSWR=10
      GOTO(771,772,773,774,70,70,70,70,70,70)NANSWR
      GOTO 77
C
C     * show a decision table *
  771 CALL GETTAB(NTAB)
      CALL SHOW(NTAB)
      IF(LASTRL(1,NTAB).NE.0)THEN
C        * give number of rule applying when table last executed *
         WRITE(*,117)LASTRL(1,NTAB)
      ELSE
         WRITE(*,118)
      ENDIF
      GOTO 70
C
C     * view the current value of a variable *
C     * get name and address *
  772 CALL GETVAR(NADRES)
      IF(NADRES.GT.0)THEN
C        * valid variable name *
         READ(12,208,REC=NADRES)UNITS,NAME
         IF(NPRD(1,NADRES).EQ.0)THEN
C           * var is undefined - is it a DCV? *
            READ(12,206,REC=NADRES)NEQN,NTABD
            IF(NEQN.EQ.0.AND.NTABD.EQ.0)THEN
C              * var a DCV ie NPRD reset to 0 after each run *
               WRITE(*,115)RCASE(1,NADRES),UNITS
            ELSE
C              * not a DCV *
               WRITE(*,116)NAME
            ENDIF
         ELSE
            WRITE(*,125)NAME,RCASE(1,NADRES),UNITS
         ENDIF
      ENDIF
      GOTO 70
C
C     * view all equations associated with a variable *
  773 CALL GETVAR(NADRES)
      NPREVS=0
      IF(NADRES.NE.0)THEN
```

```
C        * variable known *
         READ(12,206,REC=NADRES)NEQN,NTABD
         IF(NEQN.GT.0)THEN
C           * list all equations *
 7731       NPREVS=NEQN
            READ(15,209,REC=NPREVS)NEQN,EQUATN
            IF(EQUATN.NE.'      ')THEN
               WRITE(*,129)NPREVS,EQUATN
            ENDIF
            IF(NEQN.NE.0)THEN
               GOTO 7731
            END IF
         ENDIF
         WRITE(*,131)
         READ(*,207)NREPLY
      ENDIF
      GOTO 70
C
C     * display a look-up table entry *
  774 CALL GETLK(LKNAME,LKDIM,NLKUP,LKNO)
C     * locate position on axes *
      CALL GETXYZ(LKNO,LKAXIS,NCOORD,LKDIM)
C     * locate data element postion in RLKDAT() *
      CALL LKPOSN(LKNO,LKDIM,LKSIZE,LKPTR,NCOORD,LOCATE)
      WRITE(*,119)RLKDAT(LOCATE)
      GOTO 70
C
C     * return to main menu - give reminder to save data *
   80 WRITE(*,120)
      READ(*,207)NREPLY
      IF(NREPLY.EQ.'Y'.OR.NREPLY.EQ.'y')THEN
         RETURN
      ELSE
         GOTO 70
      ENDIF
C
C
  101 FORMAT(//' RUN MENU'/' --------'/1X/' Do you wish to -'
     +  /1x/' 1  select and run a program'/' 2  re-run the last program'
     + /' 3  select a case study '/' 4  save current case study'
     + /' 5  change variable status to query/don''t query each run'
     + /' 6  preset (override) the value of a variable normally '
     + /'    calculated by the program. (Resets after next run)'
     + /'    Select your case study BEFORE using override.'
     + /' 7  view a decision table'/8X,' or the current value of a '
     +  'variable'/8X,' or all equations associated with a variable'
     + /'         or a look-up table entry'
     + /' 0  return to main menu'//' :: enter number of choice - '$)
  105 FORMAT(/1X/' A list of case studies will be displayed. A copy of '
     + /' the case study you selected will be copied into the work area'
     + /' The original data remains intact and will only be modified if'
     + /' you choose to overwrite it later. (menu option 4)'//)
  106 FORMAT(' case ',I1,5x,A)
```

```
107 FORMAT(/1X/' :: enter case number of your choice,'
   +          /'    (select 1 if you wish to start from scratch) - '$)
108 FORMAT(/1X/' :: enter case number data is to be filed under '
   +' (0 to abort) - '$)
109 FORMAT(/1X' Filing of case study data aborted.')
110 FORMAT(' :: enter a title for case study ',I1,' - '$)
111 FORMAT(/1X/' :: enter a set value for this variable '
   +/'    (up to 12 digits, including a compulsory decimal point and'
   +/'    optional 2 digit exponent)  '$)
112 FORMAT(/1X/' Note that you may vary the status of this variable '
   +/' from one case study to another ')
113 FORMAT(/1X/' WARNING - this proceedure does not alter the value
   + of the variables '
   +/'           from which the overridden variable would normally be
   + calculated.')
114 FORMAT(/1X/' NOTE : this overriding will apply for only one run')
115 FORMAT(/1X/' Variable is reset each run. Current value is ',
   +       3P1E12.2E2,' ',A)
116 FORMAT(/1X/' Value of ',A,' has not yet been calculated ')
117 FORMAT(/1X/' Last rule appling was rule number ',I2)
118 FORMAT(/1X/' This table has not yet been run - no rule applies')
119 FORMAT(/1X/' Value of look-up table entry is ',3P1E12.2E2)
120 FORMAT(/1X/' REMINDER - save your current case study (option 4)'
   +          /'            if it is required, before returning.'
   +     /1x/' :: do you wish to return to the main menu?  (Y/N)- '$)
121 FORMAT(/1X/1X/' :: do you want the values of designer controlled
   + variables '/' written to the output file?  (Y/N) - '$)
122 FORMAT(1X,A,' = ',3P1E12.2E2,' ',A)
123 FORMAT(' WARNING : the value of ',A,' was not calculated during '
   +      /'           this run and the preset value '
   +      /'           of ',3P1E12.2E2,' ',A,' was used.')
124 FORMAT(/' The following are the values of the variables set
   + for this run :- ')
125 FORMAT(/1X/' ',A,' = ',3P1E12.2E2,' ',A)
126 FORMAT(/' Do you wish to DELETE the output from this LATEST'
   +       /' AND any PREVIOUS runs  (Y/N) - '$)
127 FORMAT(//' ARE YOU SURE ?? (Y/N) - '$)
128 FORMAT(/' Is the value of the
   + variable to be queried each run ? (Y/N) - '$)
129 FORMAT(4X,I4,3X,A)
130 FORMAT(//' VIEW MENU'/1X'  ---------'/1X/' Do you wish to -'
   + /' 1  view a decision table'
   + /' 2  view the current value of a variable'
   + /' 3  view all equations associated with a variable'
   + /' 4  view a look-up table entry'
   + /' 0  return to previous menu'
   +//' :: enter number of choice - '$)
131 FORMAT(//' press <RETURN> to continue .... '$)
201 FORMAT(I1)
202 FORMAT(I2)
205 FORMAT(E12.2E2)
206 FORMAT(I4,1X,I4)
207 FORMAT(A)
208 FORMAT(10X,A,1X,A)
```

```
209 FORMAT(I2,A)
901 FORMAT(/1X/' ERROR - no case study currently in use.')
902 FORMAT(/1X/' ERROR - this variable is not designer controlled.'
   +          /'         To override its normally calculated value'
   +          /'         use menu option 6')
    END
```

```
      SUBROUTINE DETAB(NTAB,NEDIT,NWHICH)
C*****************************************************************
C   PURPOSE :   interactive entry and modification of decision tables.
C   ROUTINES CALLED :
C               ENCODE
C               INPUTS
C               XINDEX
C   ARGUMENTS :
C               NTAB    no. of decision table to be entered/modified.
C               NEDIT   if >0 indicates which aspect of table is to
C                       modified.
C               NWHICH  indicates which condition, action or rule is
C                       to be modified.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C               VAR     holds name to be used by INPUTS.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  MIDARG=0 on return from ENCODE implies input error.
C              The dependence list of a condition stub consists of
C              decision table numbers & NDEPN(1)=0, NDEPN(2)=list lgth.
C*****************************************************************
      CHARACTER TABLE,STRING(80),VAR(80),TEST(35,5),ENTRY,
     +NREPLY,STUB(50,20),EQUATN*70
      CHARACTER*30 TITLE(100),MESAGE(50),UNITS*10,NAME*20
      INTEGER NDEPN(50),NSORCE(50)
C
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +             NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK3/STRING,TEST
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
      COMMON/BLOK8/NDCV(100,30)
C
      ICOND=1
      NDCOND=0
      IACT=1
      NDACT=0
      NUMROW=0
      NUMRUL=0
C
      IF(NEDIT.GT.0)THEN
C        * not a new table, set up stubs *
C        * condition stubs *
         IF(NEDIT.EQ.1.AND.NWHICH.EQ.NCOND(NTAB))THEN
C           * last condition not yet entered *
            NEND51=NCOND(NTAB)-1
         ELSE
            NEND51=NCOND(NTAB)
         ENDIF
         DO 51 KCOND=1,NEND51
            NREC=NSTORE(NTAB,KCOND)
            READ(12,108,REC=NREC)(STUB(KCOND,J),J=1,20)
```

```
   51    CONTINUE
C        * action stubs *
         IF(NEDIT.EQ.2.AND.NWHICH.EQ.NACT(NTAB))THEN
C           * last action not yet entered *
            NEND52=NACT(NTAB)-1
         ELSE
            NEND52=NACT(NTAB)
         ENDIF
         DO 52 KACT=1,NEND52
            KROW=NCOND(NTAB)+KACT
            IF(NTYPE(NTAB,KACT).EQ.1.OR.NTYPE(NTAB,KACT).EQ.2)THEN
               NREC=NSTORE(NTAB,KROW)
               READ(12,108,REC=NREC)(STUB(KROW,J),J=1,20)
            ENDIF
   52    CONTINUE
      ENDIF
      GOTO(801,802,803)NEDIT
C
C     * if NEDIT=0 then new table required *
C     * give input instructions *
   70 WRITE(*,10)TITLE(NTAB)
      READ(*,11)NCOND(NTAB)
      WRITE(*,12)
      READ(*,11)NACT(NTAB)
      IF((NCOND(NTAB)+NACT(NTAB)).GT.50)THEN
         WRITE(*,31)
         GOTO 70
      END IF
   77 WRITE(*,13)
      READ(*,11)NRULE(NTAB)
      IF(NRULE(NTAB).GT.50)THEN
         WRITE(*,32)
         GOTO 77
      END IF
      NDCOND=NCOND(NTAB)
      NDACT=NACT(NTAB)
      NDRULE=NRULE(NTAB)
      GOTO 804
C
C     * condition to be re-entered *
  801 ICOND=NWHICH
      NDCOND=NWHICH
      NUMROW=NDCOND
      GOTO 804
C
C     * action to be re-entered *
  802 IACT=NWHICH
      NDACT=NWHICH
      NUMROW=NCOND(NTAB)+NDACT
      GOTO 804
C
C     * rule to be re-entered *
  803 NUMRUL=NWHICH
C
```

```
  804 CONTINUE
C     * input of condition stub *
      WRITE(*,33)
      DO 41 KCOND=ICOND,NDCOND
C        *ask for condition *
  901    WRITE(*,14)KCOND
         READ(*,15,ERR=901)(STRING(J),J=1,20)
         DO 471 K=1,20
            STUB(KCOND,K)=STRING(K)
  471    CONTINUE
C        * following arguments required because of dual use of INDEX *
         NARG1=1
C        * so new name will be assigned a storage location *
         NPT1=1
         NPT2=20
C        * store stub as a name *
         CALL XINDEX(NARG1,STRING,NPT1,NPT2,NADRES,NFLAG)
         IF(NFLAG.EQ.0)THEN
C           * stub previously unknown *
            UNITS='          '
            NPREVS=0
            NUMEQN=NUMEQN+1
C           * process stub as equation *
            LENGTH=20
            CALL ENCODE(NPREVS,NUMEQN,MIDEQN,LENGTH,NADRES)
            IF(MIDEQN.EQ.0)GOTO 901
            IF(MIDEQN.EQ.1)THEN
C              * DCV, no equation required *
               NEQN=0
               NUMEQN=NUMEQN-1
            ELSE
               NEQN=NUMEQN
            END IF
            NDEPN(1)=0
            NDEPN(2)=1
C           * record dependent table *
            NDEPN(3)=NTAB
         ELSE
C           * stub previously known *
C           * add NTAB to list of dependent tables *
C           * read in existing table list *
            READ(12,341,REC=NADRES)NEQN,NTABD,UNITS,
     +         (STRING(J),J=1,20),(NDEPN(K),K=1,50),
     +         (NSORCE(L),L=1,50)
            NDEPN(2)=NDEPN(2)+1
            NDEPN(NDEPN(2)+2)=NTAB
         ENDIF
         NTABD=0
         WRITE(12,341,REC=NADRES)NEQN,NTABD,UNITS,
     +      (STRING(J),J=1,20),(NDEPN(K),K=1,50),(NSORCE(L),L=1,50)
         NSTORE(NTAB,KCOND)=NADRES
   41 CONTINUE
C
```

```
C     * entry of action stubs. action may be the calculation or
C     * display of the value of a variable, decision table
C     * execution or the display of a message *
C
C     * give input instructions *
      WRITE(*,16)
      DO 42 KACT=IACT,NDACT
         KROW=NCOND(NTAB)+KACT
C        *request action type *
   71    WRITE(*,17)KACT
         READ(*,18)NANSWR
         GOTO(72,73,74,75)NANSWR
         GOTO 71
C
C        * calculation of variable *
   72    WRITE(*,19)
         READ(*,15,ERR=72)(STRING(J),J=1,20)
         DO 472 K=1,20
            STUB(KROW,K)=STRING(K)
            VAR(K)=STRING(K)
  472    CONTINUE
         CALL XINDEX(NARG1,STRING,NPT1,NPT2,NADRES,NFLAG)
         NPREVS=0
         READ(12,34,REC=NADRES)NEQN,NTABD,UNITS,NAME,NDEPN(1),
     +      (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
         IF(NEQN.EQ.0.AND.NTABD.EQ.0)THEN
C           * request units for variable (subsequent input of
C           * new eqn would mean DESPGM wouldn't request units) *
            WRITE(*,106)(STRING(J),J=1,20)
            READ(*,24)UNITS
         END IF
         IF(NFLAG.EQ.1)THEN
C           * variable previously known *
            NREC=NADRES
            IF(NEQN.GT.0)THEN
C              * list all existing equations for this variable *
               WRITE(*,36)(STRING(J),J=1,20)
   79          NPREVS=NEQN
               READ(15,35,REC=NPREVS)NEQN,EQUATN
               IF(EQUATN.NE.'      ')THEN
                  WRITE(*,37)NPREVS,EQUATN
               ENDIF
               IF(NEQN.NE.0)THEN
                  GOTO 79
               END IF
               WRITE(*,38)
               READ(*,103)NEQN
            END IF
         ELSE
C           * variable previously unknown *
            NEQN=0
         END IF
         IF(NEQN.EQ.0)THEN
C           * new equation required *
            NUMEQN=NUMEQN+1
```

```
            NEQN=NUMEQN
  902       CALL INPUTS(VAR)
            LENGTH=70
            CALL ENCODE(NPREVS,NUMEQN,MIDEQN,LENGTH,NADRES)
            IF(MIDEQN.EQ.0)THEN
C              * error in equation given to ENCODE - retry *
               GOTO 902
            ENDIF
            IF(NPREVS.EQ.0)THEN
               WRITE(12,34,REC=NADRES)NUMEQN,NTABD,UNITS,NAME,
     +            (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
            END IF
         END IF
         NTYPE(NTAB,KACT)=1
         NSTORE(NTAB,KROW)=NADRES
         METHOD(NTAB,KACT)=NEQN
         GOTO 42
C
C        * display value of a variable *
   73    WRITE(*,19)
         READ(*,15,ERR=73)(STRING(J),J=1,20)
         DO 473 K=1,20
            STUB(KROW,K)=STRING(K)
  473    CONTINUE
         CALL XINDEX(NARG1,STRING,NPT1,NPT2,NADRES,NFLAG)
         NTYPE(NTAB,KACT)=2
         NSTORE(NTAB,KROW)=NADRES
         METHOD(NTAB,KACT)=0
         GOTO 42
C
C        * execute a decision table *
C        * display list of existing tables *
   74    WRITE(*,22)
         DO 43 KTAB=1,NDETAB
C           * check whether table is active *
            IF(NCOND(KTAB).EQ.0)GOTO 43
            WRITE(*,20)KTAB,TITLE(KTAB)
   43    CONTINUE
         WRITE(*,21)
         READ(*,11)NANSWR
         IF(NANSWR.GT.NDETAB)GOTO 74
         IF(NANSWR.NE.0)THEN
            METHOD(NTAB,KACT)=NANSWR
         ELSE
            NDETAB=NDETAB+1
            METHOD(NTAB,KACT)=NDETAB
            WRITE(*,23)
            READ(*,24)TITLE(NDETAB)
         END IF
         NTYPE(NTAB,KACT)=3
         NSTORE(NTAB,KROW)=0
         GOTO 42
C
```

```
C        * display a message *
C        * display list of existing messages *
   75    DO 44 KMESGE=1,NMESGE
            WRITE(*,20)KMESGE,MESAGE(KMESGE)
   44    CONTINUE
         WRITE(*,25)
         READ(*,11)NANSWR
         IF(NANSWR.GT.NMESGE)GOTO 75
         IF(NANSWR.NE.0)THEN
            METHOD(NTAB,KACT)=NANSWR
         ELSE
            NMESGE=NMESGE+1
            METHOD(NTAB,KACT)=NMESGE
            WRITE(*,26)
            READ(*,24)MESAGE(NMESGE)
         END IF
         NTYPE(NTAB,KACT)=4
         NSTORE(NTAB,KROW)=0
   42 CONTINUE
C
C     * input of table rules *
      NROW=NCOND(NTAB)+NACT(NTAB)
      DO 46 KRULE=1,NRULE(NTAB)
         WRITE(*,27)KRULE
         KKRULE=KRULE-1
         DO 45 KROW=1,NCOND(NTAB)
   76       WRITE(*,29)(STUB(KROW,J),J=1,20)
            IF(NEDIT.EQ.0 .OR. KROW.EQ.NUMROW .OR. KRULE.EQ.NUMRUL)THEN
               DO 481 KENTRY=1,KKRULE
                  WRITE(*,104)TABLE(NTAB,KROW,KENTRY)
  481          CONTINUE
               READ(*,30)ENTRY
C              * convert to upper case *
               IF(ENTRY.EQ.'y')ENTRY='Y'
               IF(ENTRY.EQ.'n')ENTRY='N'
               IF(ENTRY.NE.'Y'.AND.ENTRY.NE.'N'.AND.ENTRY.NE.'-')THEN
                  WRITE(*,91)
                  GOTO 76
               ELSE
                  TABLE(NTAB,KROW,KRULE)=ENTRY
               END IF
            ELSE
               DO 482 KENTRY=1,KRULE
                  WRITE(*,104)TABLE(NTAB,KROW,KENTRY)
  482          CONTINUE
            END IF
   45    CONTINUE
C        * separate condition and action entries *
         IF(NEDIT.GT.0)WRITE(*,109)
         DO 47 K=1,KRULE
            WRITE(*,281)
   47    CONTINUE
         WRITE(*,28)
C        * action entries *
         DO 451 KACT=1,NACT(NTAB)
```

```
            KROW=NCOND(NTAB)+KACT
  770       GOTO(771,772,773,774)NTYPE(NTAB,KACT)
C           * variable calculation *
  771       WRITE(*,100)(STUB(KROW,J),J=1,20)
            GOTO 775
C           * variable display *
  772       WRITE(*,105)(STUB(KROW,J),J=1,20)
            GOTO 775
C           * execute a decision table *
  773       WRITE(*,101)TITLE(METHOD(NTAB,KACT))
            GOTO 775
C           * display a message *
  774       WRITE(*,102)MESAGE(METHOD(NTAB,KACT))
  775       IF(NEDIT.EQ.0 .OR. KROW.EQ.NUMROW .OR. KRULE.EQ.NUMRUL)THEN
               DO 484 KENTRY=1,KKRULE
                  WRITE(*,104)TABLE(NTAB,KROW,KENTRY)
  484             CONTINUE
               READ(*,30)ENTRY
               IF(ENTRY.NE.'X'.AND.ENTRY.NE.' ')THEN
                  WRITE(*,91)
                  GOTO 770
               ELSE
                  TABLE(NTAB,KROW,KRULE)=ENTRY
               END IF
            ELSE
               DO 483 KENTRY=1,KRULE
                  WRITE(*,104)TABLE(NTAB,KROW,KENTRY)
  483             CONTINUE
            END IF
  451    CONTINUE
   46 CONTINUE
C      * decision table input completed *
C
       RETURN
C
    10 FORMAT(//1X,'Decision table  ',A,
      +     /1X,'is now to be entered. Conditions and actions'
      +     /1X,'may not exceed 50 in total.'
      + /1X/1X,':: enter number of conditions - '$)
    11 FORMAT(I2)
    12 FORMAT(' :: enter number of actions    - '$)
    13 FORMAT(' :: enter number of rules      - '$)
    14 FORMAT(/' :: enter condition number ',I2,
      +       ', no more than 20 characters long -')
    15 FORMAT(20A1)
    16 FORMAT(/1X/1X/1X,'Decision table actions are now to be entered.')
    17 FORMAT(/1X/1X,'Is action number ',I2,' to consist of :-'
      +/7x,'1  calculation of a variable'
      +/7x,'2  displaying the value of a variable'
      +/7x,'3  executing a decision table'
      +/7x,'4  displaying a message'
      +/1X,/1X,':: enter action type number - '$)
    18 FORMAT(I1)
    19 FORMAT(/' :: enter variable name, up to 20 characters long -')
```

```
 20 FORMAT(10X,I2,7X,A)
 21 FORMAT(/1X,/1X,':: enter number of table, '
   +       '0 if one not yet listed - '$)
 22 FORMAT(' :: the following are existing or reserved tables :-'
   +/1X,/4X,'TABLE NUMBER',8X,'TITLE'/1X)
 23 FORMAT(' :: enter title of table, up to 30 characters long - ')
 24 FORMAT(A)
 25 FORMAT(/1X,':: enter number of message, '
   +       '0 if one not yet listed - '$)
 26 FORMAT(' :: enter message, up to 30 characters long -')
 27 FORMAT(/1X,'Entry of rule ',I2
   +/1X,'  Condition entries may be Y,N or - '
   +/1X,'  Action entries may be X or blank.'
   +/1X,/1X,'Make entry when cursor stops at each row.'/1X,/1X)
 28 FORMAT('+',30('='))
281 FORMAT('+='$)
 29 FORMAT(1X,'     ',20A1,'? || '$)
 30 FORMAT(A1)
 31 FORMAT(' ERROR. Total number of conditions and actions > 50')
 32 FORMAT(' ERROR. Total number of rules exceeds 50')
 33 FORMAT(' Entry of decision table conditions.'/1X
   +/1X,'Conditions must not be more than 20 characters long and'
   +/1X,'must not contain operators other than < > or ='/1X)
 34 FORMAT(I4,1X,I4,1X,A10,1X,A20,50I4,50I4)
341 FORMAT(I4,1X,I4,1X,A10,1X,20A1,50I4,50I4)
 35 FORMAT(I2,A)
 36 FORMAT(' The following are the existing equations for the '
   +/1X'calculation of ',20A1,/1X)
 37 FORMAT(4X,I4,1X,A)
 38 FORMAT(/1X/1X,':: enter desired equation number, 0 if a new
   +equation is required - '$)
 39 FORMAT(11X,A)
 91 FORMAT(' ERROR - invalid entry, try again'/1X)
100 FORMAT(1X,'calc ',20A1,'  || '$)
101 FORMAT(1X,'do   ',A20,'  || '$)
102 FORMAT(1X,'say  ',A20,'  || '$)
103 FORMAT(I2)
104 FORMAT('+',A1$)
105 FORMAT(1X,'show ',20A1,'  || '$)
106 FORMAT(/' :: enter units for ',20A1
   +/1X,'   up to 10 characters or press <RETURN> if dimensionless -')
107 FORMAT(A10,1X,20A1)
108 FORMAT(21X,20A1)
109 FORMAT(/)
    END
```

```
      SUBROUTINE ENCODE(NPREVS,NEQN,MIDARG,LENGTH,LOCATE)
C*****************************************************************
C   PURPOSE :  encoding of equations into a numeric form.
C   METHODS :  Original equation is scanned for variable and look-up
C              table names, and standard operators and functions listed
C              in a table of keywords.
C   ROUTINES CALLED :
C              DEPLST
C              LOOK
C              RPN
C              XINDEX
C   ARGUMENTS :
C              NPREVS  no. of previous equation in linked list of eqns.
C              NEQN    no. of eqn being encoded.
C              MIDARG  no. of entries in encoded eqn.
C              LENGTH  max. length of input eqn.
C              LOCATE  address of eqn left-hand side.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C              MIDEQN  holds encoded eqn.
C              NUMARG, NUMLK  monitor parameters of referenced look-ups.
C   LOCAL VARIABLES :
C              NARG    points to current position in scanning input eqn.
C              NCODE   keyword code of current eqn element.
C              LAST    keyword code of last eqn element.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  MIDEQN is converted to Rev. Polish form by RPN.
C              ENCODE adds a ) to input eqns to force the processing
C              of the last eqn element.
C              LAST=33 indicates look-up table or variable name.
C              List of keywords and checking array is in KEYWRD.dat.
C              MIDARG=0 indicates error condition.
C*****************************************************************
      CHARACTER TEST(35,5),STRING(80),SPACE*3
      INTEGER MIDEQN(30),HOLD(80)
      COMMON/BLOK2/ISIZE(35),ICODE(35),ICHEC(35,35),NKEYWD
      COMMON/BLOK3/STRING,TEST
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
C     * add ')' to end of eqn. to force processing of last variable name *
      LENGTH=LENGTH+1
      STRING(LENGTH)=')'
C
      NERROR=0
      LAST=1
      MIDEQN(1)=0
      NARG=1
      NSTART=1
      SPACE='   '
      NUMARG=0
      LKNO=0
      LKSCAN=0
```

```
C
   71 IF(NARG.GT.LENGTH)THEN
C        * remove end of equation ')' *
         STRING(LENGTH)=' '
C        * write original equation to file *
         IF(NEQN.EQ.NUMEQN)THEN
C     * a new eqn has been encoded *
            N=0
         ELSE
C     * a modified eqn has been encoded *
            READ(15,12,REC=NEQN)N
         ENDIF
         WRITE(15,10,REC=NEQN)N,(STRING(J),J=1,LENGTH)
         IF(NPREVS.NE.0)THEN
C           * form linked list for multi-equationed variables *
            READ(15,10,REC=NPREVS)N,(STRING(J),J=1,80),
     +                              (HOLD(K),K=1,HOLD(1)+1)
            WRITE(15,10,REC=NPREVS)NEQN,(STRING(J),J=1,80),
     +                              (HOLD(K),K=1,HOLD(1)+1)
         END IF
         MIDARG=MIDEQN(1)-1
         CALL RPN(MIDEQN,NEQN)
         CALL DEPLST(LOCATE,NEQN)
C
         RETURN
C
      END IF
C     * CHECK FOR BLANKS *
      IF(STRING(NARG).EQ.' ')THEN
         NARG=NARG+1
         GOTO 71
      END IF
C
C
C     * compare STRING with table of standard keywords (SIN, COS etc) *
      DO 42 KROW=1,NKEYWD
         DO 41 KCOL=1,ISIZE(KROW)
C           * ISIZE is length of entry in table *
            IF(STRING(NARG+KCOL-1).NE.TEST(KROW,KCOL))GOTO 42
   41    CONTINUE
C        * normal exit from loop 41 implies successful match *
C        * keyword may mark end of constant, variable name
C        * or look-up table name *
C
C        * 'PI' may occur as part of a variable name *
         IF(ICODE(KROW).EQ.20.AND.LAST.EQ.1)THEN
C           * letters 'PI' encountered at beginning of eqn *
            IF(STRING(NARG+2).GE.'A'.AND.STRING(NARG+2).LE.'Z')THEN
C              * 'PI' first letters of a name *
               GOTO 73
            ENDIF
         ELSEIF(ICODE(KROW).EQ.20.AND.LAST.EQ.33)THEN
C           * letters 'PI' are embedded in a character string *
```

```
            GOTO 73
         ENDIF
C
C        * TRUE, FALSE, YES, NO may occur as part of a var. name *
C        * Y, N, T, F not allowed as logicals - can't detect whether
C        * logicals or variable names *
         IF(ICODE(KROW).EQ.22.OR.ICODE(KROW).EQ.23)THEN
            IF(LAST.EQ.1)THEN
C              * beginning of equation *
               IF(STRING(NARG+2).GE.'A'.AND.STRING(NARG+2).LE.'Z')THEN
C                 * first letters of a name *
                  GOTO 73
               ENDIF
            ELSEIF(LAST.EQ.33)THEN
C              * letters are embedded in a character string *
               GOTO 73
            ENDIF
         ENDIF
C
         NCODE=ICODE(KROW)
C
         IF(LKSCAN.EQ.1.AND.NCODE.NE.11.AND.NCODE.NE.4)THEN
C           * ERROR - operator within look-up table arguments *
            WRITE(*,97)
            NERROR=1
            GOTO 79
         ENDIF
C
         IF(LAST.EQ.33.AND.NCODE.LE.20)THEN
C           * element marks end of variable or look-up table name *
            IF(NCODE.EQ.10)THEN
C              * open ( marks end of look-up table name *
               LKSCAN=1
               NEND=NARG-1
               CALL LOOK(NSTART,NEND,STRING,NADRES)
               IF(NADRES.EQ.0)THEN
C                 * error condition *
                  MIDARG=0
                  RETURN
               ELSE
C                 * note table number *
                  LKNO=NADRES-35
               ENDIF
            ELSE
C              * variable name *
               NEND=NARG-1
               CALL XINDEX(LAST,STRING,NSTART,NEND,NADRES,NFLAG)
            ENDIF
            MIDEQN(1)=MIDEQN(1)+1
            MIDEQN(MIDEQN(1)+1)=NADRES
C
            IF(NCODE.EQ.11.AND.LKSCAN.EQ.1)THEN
C              * close bkt marks end of look-up table arguments *
               NUMARG=NUMARG+1
```

```
C              * if table already exists is the no. of arg's correct? *
               IF(LKPTR(LKNO).EQ.0)THEN
C                 * first time table encountered - set parameters *
                  LKDIM(LKNO)=NUMARG
               ELSE
                  IF(LKDIM(LKNO).NE.NUMARG)THEN
C                    * error - incorrect no. arguments *
                     WRITE(*,95)LKDIM(LKNO),NUMARG
                     NERROR=1
                     GOTO 79
                  ELSE
C                    * all OK *
                     LKSCAN=0
                     NUMARG=0
                     LKNO=0
                  ENDIF
               ENDIF
            ENDIF
         END IF
C
C        * check for commas separating look-up table arguments *
         IF(NCODE.EQ.4.AND.LKSCAN.EQ.1)THEN
            NUMARG=NUMARG+1
         ENDIF
C
   70    IF(NCODE.GE.21.AND.NCODE.LE.24)THEN
C           * digit in input *
            IF(LAST.NE.33)THEN
C              * scanning a leading digit - error *
               WRITE(*,94)
               NERROR=1
               GOTO 79
            ELSE
C              * digit in variable or look-up table name *
               GOTO 73
            END IF
         ELSE
C           * current eqn element is not a constant *
C           * insert code into equation *
            MIDEQN(1)=MIDEQN(1)+1
            MIDEQN(MIDEQN(1)+1)=NCODE
            IF(ICODE(KROW).LE.19.AND.ICODE(KROW).GE.12)THEN
C              * add ( to library functions *
               MIDEQN(1)=MIDEQN(1)+1
               MIDEQN(MIDEQN(1)+1)=10
               LAST=10
            END IF
         END IF
C
C        * step *
         NARG=NARG+ISIZE(KROW)
         NSTART=NARG
         NSFLAG=0
         GOTO 72
```

```
C
   42 CONTINUE
C
C     * normal exit implies variable or look-up table name *
C     * name will end at '(',')', equation end or operator *
   73 NARG=NARG+1
      NCODE=33
      NSFLAG=1
*    72 IF(LAST.EQ.1)THEN
*C        * NARG is at first element of equation *
*         LAST=NCODE
*       ENDIF
   72 IF(ICHEC(NCODE,LAST).NE.1)THEN
C        * incompatability in input equation *
         IF(NSFLAG.EQ.1)THEN
            NARG=NARG-1
         ENDIF
         WRITE(*,91)
         NERROR=1
      ELSE
         LAST=NCODE
      END IF
   79 IF(NERROR.EQ.1)THEN
C        * put ? above source of error, write blanks *
         DO 50 K50=1,NARG
            WRITE(*,96)
   50    CONTINUE
C        * write ? *
         WRITE(*,92)
         WRITE(*,93)(STRING(J),J=1,NARG),SPACE,(STRING(K),K=(NARG+1),30)
         MIDARG=0
C        * indicate error to DETAB.for or DESPGM.for *
         NADRES=0
         RETURN
      ENDIF
      GOTO 71
C
   10 FORMAT(I2,80A1,80I4)
   11 FORMAT(5(I1,1X,E12.2E2))
   12 FORMAT(I2)
   91 FORMAT(/1X,/1X,'ERROR in input string'//)
   92 FORMAT('+?')
C    92 FORMAT('?')                                          PC line !!
   93 FORMAT(1X,83A1)
   94 FORMAT(/1X,/1X,'ERROR - digit as leading character in name'//)
   95 FORMAT(/1X/' ERROR - incorrect number of look-up table arguments'
     +          /'         correct number is ',I2
     +          /'         number entered is ',I2)
   96 FORMAT('+ '$)
C    96 FORMAT(' '$)                                         PC line !!
   97 FORMAT(//' ERROR - operators not permitted within look-up '
     +         'table arguments')
      END
```

```
      SUBROUTINE EQN(MID)
C*************************************************************************
C   PURPOSE : puts elements from MIDEQN into NRPNEQ under control of RPN
C   ARGUMENTS :
C             MID  element to be entered into NRPNEQ.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      INTEGER MIDEQN(30)
      COMMON/BLOK9/NRPNEQ(50)
      NRPNEQ(1)=NRPNEQ(1)+1
      NRPNEQ(NRPNEQ(1)+1)=MID
      RETURN
      END
```

```
      SUBROUTINE GETLK(LKNAME,LKDIM,NLKUP,LKNO)
C*****************************************************************
C   PURPOSE :  Interactive selection of a look-up table from a list of
C              all tables.
C   ARGUMENTS :
C              LKNAME  look-up table names.
C              LKDIM   no. of dimensions to each table.
C              NLKUP   no. of look-up tables.
C              LKNO    no. of table selected.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*****************************************************************
      CHARACTER*20 LKNAME(15)
      INTEGER LKDIM(15)
   71 WRITE(*,119)
      DO 46 LKNO=1,NLKUP
         IF(LKDIM(LKNO).GT.0)THEN
C           * table is active, give name *
            WRITE(*,120)LKNO,LKNAME(LKNO)
         ENDIF
   46 CONTINUE
C     * select look-up table *
      WRITE(*,121)
      READ(*,202)LKNO
      IF(LKNO.GT.15.OR.LKNO.LT.1.OR.LKDIM(LKNO).EQ.0)THEN
C        * inactive or invalid table nominated, repeat process *
         WRITE(*,91)
         GOTO 71
      ENDIF
      RETURN
C
   91 FORMAT(/1X/' ERROR - inactive table nominated. Try again.')
  119 FORMAT(/1X/' Active look-up tables are :-'/1x/1x)
  120 FORMAT(' Look-up table number ',I2,5x,A)
  121 FORMAT(/1X/' :: enter number of table - '$)
  202 FORMAT(I2)
      END
```

```
      SUBROUTINE GETTAB(NTAB)
C*************************************************************************
C   PURPOSE : selection of a decision table from a list of tables.
C   ARGUMENTS :  NTAB  number of decision table selected.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C   LOCAL VARIABLES :
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      CHARACTER*30 TITLE(100)
C
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
C
      WRITE(*,102)
      DO 41 KTAB=1,NDETAB
C        * check whether table is active *
         IF(NCOND(NTAB).GT.0)THEN
            WRITE(*,103)KTAB,TITLE(KTAB)
         ENDIF
   41 CONTINUE
      WRITE(*,104)
      READ(*,202)NTAB
      RETURN
C
  102 FORMAT(//' The following is a list of all decision tables :-'/)
  103 FORMAT(' table ',I2,5X,A)
  104 FORMAT(//' :: enter number of decision table - '$)
  202 FORMAT(I2)
      END
```

```
      SUBROUTINE GETVAR(NADRES)
C*************************************************************************
C   PURPOSE :  to request a variable name from the user & return the
C              corresponding address to the calling program.
C   ROUTINES CALLED :  XINDEX
C   ARGUMENTS :  NADRES  address of variable specified by user.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  NADRES=0 indicates invalid variable name entered.
C*************************************************************************
      CHARACTER STRING(20)
C
      WRITE(*,101)
      READ(*,201)(STRING(J),J=1,20)
C     * NARG1=0 so that a mis-spelt name is not entered onto file *
      NARG1=0
      NPT1=1
      NPT2=20
      CALL XINDEX(NARG1,STRING,NPT1,NPT2,NADRES,NFLAG)
      RETURN
C
  101 FORMAT(/' :: enter name of variable - '$)
  201 FORMAT(20A1)
      END
```

```
      SUBROUTINE GETXYZ(LKNO,LKAXIS,NCOORD,LKDIM)
C*************************************************************************
C   PURPOSE :  Interactive entry of coordinates of data element in
C              a look-up table.
C   ARGUMENTS :
C              LKNO    look-up table number.
C              LKAXIS  look-up table axis labels.
C              NCOORD  coordinate on each axis of look-up table.
C              LKDIM   no. of dimensions of look-up table axis.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      INTEGER NCOORD(5),LKDIM(15)
      CHARACTER*20 LKAXIS(15,5)
C
      DO 41 KDIM=1,LKDIM(LKNO)
         WRITE(*,101)LKAXIS(LKNO,KDIM)
         READ(*,201)NCOORD(KDIM)
   41 CONTINUE
      RETURN
C
  101 FORMAT(/1X/' :: enter coordinate for ',A,' - '$)
  201 FORMAT(I2)
      END
```

```
      SUBROUTINE INPUT(LHS)
C*********************************************************************
C   PURPOSE : entry of equation right hand sides.
C   ARGUMENTS :  LHS  storage address of equation LHS
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  partner subroutines to INPUTS.
C              INPUT uses an A20 format LHS.
C*********************************************************************
      CHARACTER LHS*20,STRING(80),TEST(35,5)
      COMMON/BLOK3/STRING,TEST
      WRITE(*,10)LHS
      READ(*,11)(STRING(J),J=1,70)
      RETURN
   10 FORMAT(//' :: enter RHS of equation defining ',A
     +      /'    no more than 70 characters -')
   11 FORMAT(70A1)
      END
```

```
      SUBROUTINE INPUTS(LHS)
C*********************************************************************
C   PURPOSE : entry of equation right hand sides.
C   ARGUMENTS :  LHS  storage address of equation LHS
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  partner subroutines to INPUT.
C              INPUT uses an 20A1 format LHS.
C*********************************************************************
      CHARACTER LHS(20),STRING(80),TEST(35,5)
      COMMON/BLOK3/STRING,TEST
      WRITE(*,10)(LHS(J),J=1,20)
      READ(*,11)(STRING(J),J=1,70)
      RETURN
   10 FORMAT(//' :: enter RHS of equation defining ',20A1
     +      /'    no more than 70 characters -')
   11 FORMAT(70A1)
      END
```

```
      SUBROUTINE INSERT(LENGTH,NAME,LOCATE,NADRES)
C*************************************************************************
C   PURPOSE :  To insert new names into the directory of names stored in
C              DIRECT.dat.
C   METHODS :  If space needs to be made between adjacent name-length
C              blocks a standard 5 spaces are inserted to reduce the
C              number of shifts required.
C   ARGUMENTS :
C              LENGTH  length of name to be inserted.
C              NAME    name of new variable.
C              LOCATE  location for new name in the directory as
C                      allocated by XINDEX.
C              NADRES  storage address of variable in case studies
C                      (allocated sequentially).
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C              NSHIFT  no. of lines each block of names needs to be
C                      shifted if insufficient space unavailable.
C              NALPHA  first position occupied by each name length
C                      block in directory.
C              NOMEGA  last position occupied by each name length
C                      block in directory.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      CHARACTER HOLD*20,NAME*20,UNITS*10
      INTEGER NSHIFT(20),NALPHA(20),NOMEGA(20)
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
C
      NSHIFT(1)=0
      NREC=LENGTH
      READ(13,14,REC=NREC)NEND
      NREC=LENGTH+1
      READ(13,10,REC=NREC)NBEGIN
      IF((NBEGIN-NEND).EQ.1)THEN
C        * need to make room in directory *
C        * adjust to give 5 spaces between blocks *
C        * read space occupied by each block of names *
         DO 41 K41=1,20
            READ(13,11,REC=K41)NALPHA(K41),NOMEGA(K41)
   41    CONTINUE
C        * calculate amount of shift for each block of names *
         DO 42 K42=2,20
            NSHIFT(K42)=6-(NALPHA(K42)-NOMEGA(K42-1))+NSHIFT(K42-1)
   42    CONTINUE
C        * allow spacd for insertion of current name *
         DO 46 K46=(LENGTH+1),20
            NSHIFT(K46)=NSHIFT(K46)+1
   46    CONTINUE
C        * carry out shifts *
         DO 43 K43=2,20
C             * KEND gives block length *
```

```
            KEND=NOMEGA(22-K43)-NALPHA(22-K43)+1
            DO 44 K44=1,KEND
                NREC=NOMEGA(22-K43)+1-K44
                READ(13,12,REC=NREC)NADRES,HOLD
                NREC=NREC+NSHIFT(22-K43)
                WRITE(13,12,REC=NREC)NADRES,HOLD
 44         CONTINUE
            NREC=22-K43
            NALPHA(NREC)=NALPHA(NREC)+NSHIFT(NREC)
            NOMEGA(NREC)=NOMEGA(NREC)+NSHIFT(NREC)
            WRITE(13,11,REC=NREC)NALPHA(NREC),NOMEGA(NREC)
 43      CONTINUE
         LOCATE=LOCATE+NSHIFT(LENGTH)
      END IF

C     * add name to directory *
C     * NENTRY is number of variables currently in system *
      NENTRY=NENTRY+1
      READ(13,11,REC=LENGTH)NBEGIN,NEND
C     * make space for new variable name *
C     * KEND is block length or number of elements to be moved *
      KEND=NEND-LOCATE+1
      NREC=NEND+2
      DO 45 K45=1,KEND
         NREC=NEND+1-K45
         READ(13,12,REC=NREC)NADRES,HOLD
         NREC=NREC+1
         WRITE(13,12,REC=NREC)NADRES,HOLD
 45   CONTINUE
      NREC=NREC-1
      NADRES=NENTRY
      WRITE(13,12,REC=NREC)NADRES,NAME
      NEND=NEND+1
      WRITE(13,11,REC=LENGTH)NBEGIN,NEND
C     * write name to FIXED.dat file *
      NEQN=0
      NTAB=0
      NDEPN=0
      UNITS='          '
      WRITE(12,11,REC=NADRES)NEQN,NTAB,UNITS,NAME,NDEPN
C     * initialise case studies to 0 *
      DO 47 KCASE=1,6
         NPRD(KCASE,NADRES)=0
         RCASE(KCASE,NADRES)=0
 47   CONTINUE
      RETURN
C
 10   FORMAT(I4)
 11   FORMAT(I4,1X,I4,1X,A10,1X,A20,I4)
 12   FORMAT(I4,1X,A)
 13   FORMAT(11X,A)
 14   FORMAT(5X,I4)
      END
```

```
      SUBROUTINE LKLOAD(LKNO)
C*************************************************************************
C   PURPOSE :  to interactively load data into look-up tables.
C   METHODS :  5 nested loops reflect the max. no. of look-up table
C              dimensions.
C   ARGUMENTS : LKNO  number of look-up table to be loaded.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C   LOCAL VARIABLES :
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  Data for all look-up tables is stored in a single
C              dimension array RLKDAT.
C*************************************************************************
      CHARACTER*20 LKAXIS(15,5),LKNAME(15)
      CHARACTER*30 CASE(6)
      INTEGER NCOORD(5)
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
C
C     * initialise LKSIZE array *
      DO 43 KSIZE=1,5
         LKSIZE(LKNO,KSIZE)=1
   43 CONTINUE
C     * request look-up table axes name and length *
      WRITE(*,113)LKNAME(LKNO)
      DO 44 KAXIS=1,LKDIM(LKNO)
C        * request axis title (A20) *
         WRITE(*,114)KAXIS
         READ(*,108)LKAXIS(LKNO,KAXIS)
C        * request axis length (I2) *
         WRITE(*,115)KAXIS
         READ(*,104)LKSIZE(LKNO,KAXIS)
   44 CONTINUE
C     * load data into look-up table *
      WRITE(*,116)LKNAME(LKNO)
      LOCATE=LKNEXT-1
      LKPTR(LKNO)=LKNEXT
      DO 451 K451=1,LKSIZE(LKNO,1)
        NCOORD(1)=K451
        DO 452 K452=1,LKSIZE(LKNO,2)
          NCOORD(2)=K452
          DO 453 K453=1,LKSIZE(LKNO,3)
            NCOORD(3)=K453
            DO 454 K454=1,LKSIZE(LKNO,4)
              NCOORD(4)=K454
              DO 455 K455=1,LKSIZE(LKNO,5)
                NCOORD(5)=K455
                LOCATE=LOCATE+1
                DO 461 K461=1,LKDIM(LKNO)
                   WRITE(*,117)LKAXIS(LKNO,K461)
  461           CONTINUE
                WRITE(*,1171)
                DO 462 K462=1,LKDIM(LKNO)
                   WRITE(*,1172)NCOORD(K462)
```

```
  462           CONTINUE
                READ(*,118)RLKDAT(LOCATE)
  455         CONTINUE
  454       CONTINUE
  453     CONTINUE
  452   CONTINUE
  451 CONTINUE
      WRITE(*,119)
C     * record total no. of entries for this look-up table *
      LKTOT(LKNO)=LOCATE-LKNEXT+1
      LKNEXT=LOCATE+1
      RETURN
C
C
  104 FORMAT(I2)
  108 FORMAT(A)
  113 FORMAT(/1X/' Look-up table ',A,' is now to be processed ')
  114 FORMAT(/1X/' :: enter title of axis number ',I1,' - '$)
  115 FORMAT(//' :: enter number of entries along axis number ',
     +       I1,' - '$)
  116 FORMAT(//' Data is now to be loaded into look-up table ',A
     +        /' :: enter table entry in turn below the DATA heading,'
     +        /'    up to 12 digits including a decimal point and '
     +        /'    optional 2 digit exponent.'
     +        ///)
  117 FORMAT('+',A$)
 1171 FORMAT('+DATA'/)
 1172 FORMAT('+',I2,18(' ')$)
  118 FORMAT(E12.2E2)
  119 FORMAT(/1X/' Data for this look-up table is now fully loaded')
      END
```

```
      SUBROUTINE LKPOSN(LKNO,LKDIM,LKSIZE,LKPTR,NCOORD,LOCATE)
C*****************************************************************
C   PURPOSE :   To identify location of particular look-up table data
C               item.
C   ARGUMENTS :
C               LKNO    look-up table number.
C               LKDIM   no. of look-up table dimensions.
C               LKSIZE  length of each axis.
C               LKPTR   start location in RLKDAT of data for each
C                       look-up table.
C               NCOORD  coordinate on each axis of look-up table.
C               LOCATE  location of specific data element in RLKDAT.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  All look-up table data is stored in the one
C              dimensional array RLKDAT.
C*****************************************************************
      INTEGER LKDIM(15),LKSIZE(15,5),LKPTR(15),NCOORD(5)
C
      NTOT=0
      DO 41 KAXIS=1,LKDIM(LKNO)
         NHOLD=1
         DO 42 KAXIS2=KAXIS+1,LKDIM(LKNO)
            NHOLD=NHOLD*LKSIZE(LKNO,KAXIS2)
   42    CONTINUE
         NHOLD=NHOLD*(NCOORD(KAXIS)-1)
         NTOT=NTOT+NHOLD
   41 CONTINUE
      LOCATE=LKPTR(LKNO)+NTOT
      RETURN
      END
```

```
      SUBROUTINE LOOK(NPT1,NPT2,STRING,NADRES)
C*************************************************************************
C   PURPOSE :  To process look-up table names identified by ENCODE &
C              identify & allocate storage for new tables.
C              To check system look-up table capacity is not exceeded.
C   ARGUMENTS :
C              NPT1    start location of look-up name in STRING.
C              NPT2    end location of look-up name in STRING.
C              STRING  input eqn for encoding.
C              NADRES  returns look-up table no. to ENCODE.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL VARIABLES :
C              NLOOK   marks vacant table slot found while checking
C                      if look-up name already exists.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  Returning NADRES=0 to ENCODE indicates error condition.
C*************************************************************************
      CHARACTER STRING(80)
      CHARACTER*20 LKAXIS(15,5),LKNAME(15),NAME
      CHARACTER*30 CASE(6)
C
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
C
      NLOOK=0
C
C     * remove leading blanks from name *
      NSTART=NPT1
      NEND=NPT2
      DO 42 K42=NSTART,NPT2
         IF(STRING(K42).NE.' ')THEN
            NPT1=K42
            GOTO 72
         ENDIF
   42 CONTINUE
C     * remove trailing blanks from name *
   72 DO 43 K43=NPT1,NEND
         IF(STRING(K43).NE.' ')THEN
            NPT2=K43
         ENDIF
   43 CONTINUE
      LENGTH=NPT2-NPT1+1
C     * following read/write necessary through lack of
C     * character substring handling facility *
      WRITE(12,10,REC=2)(STRING(J),J=NPT1,NPT2)
      READ(12,14,REC=2)NAME
C
C     * does look-up table already exist? *
      DO 41 K41=1,NLKUP
         IF(LKDIM(K41).GT.0)THEN
C           * table exists, check name *
            IF(LKNAME(K41).EQ.NAME)THEN
```

```
C              * nominated table already exists *
               NADRES=K41+35
               RETURN
            ENDIF
         ELSE
C          * note vacant location for future use *
           NLOOK=K41
         ENDIF
   41 CONTINUE
C
      IF(NLKUP.EQ.15.AND.NLOOK.EQ.0)THEN
C        * maximum number of look-up tables exceeded *
         WRITE(*,91)
         NADRES=0
C        * previous line indicates error condition *
         RETURN
      ELSE IF(NLKUP.EQ.14.AND.NLOOK.EQ.0)THEN
C             * this look-up table brings system to capacity *
              WRITE(*,12)
      ENDIF
C     * look-up table does not already exist, asign location *
      IF(NLOOK.EQ.0)THEN
C        * no vacant intermediate slots *
         NLKUP=NLKUP+1
         LKUP=NLKUP
      ELSE
         LKUP=NLOOK
      ENDIF
      NADRES=LKUP+35
      LKNAME(LKUP)=NAME
      LKPTR(LKUP)=0
C     * indicate table no. LKUP is in use *
      LKDIM(LKNO)=1
C     * indicate look-up tables awaiting processing *
      LKTODO=LKTODO+1
C
      RETURN
C
   10 FORMAT(20A1)
   12 FORMAT(/1X,/1X,'WARNING - look-up table capacity of 15 has '
     +'been reached.'
     +           /1X,'          No more may be entered.')
   14 FORMAT(A)
   91 FORMAT(//' ERROR - maximum number of 15 look-up tables exceeded.')
      END
```

```
      SUBROUTINE MODIFY
C*************************************************************************
C   PURPOSE :  Interactive modification of decision tables, variables
C              and look-up tables.
C   ROUTINES CALLED :
C              DESPGM
C              SHOW
C              DETAB
C              GETVAR
C              INPUT
C              ENCODE
C              UNDEPN
C              GETLK
C              GETXYZ
C              LKPOSN
C              LKLOAD
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL VARIABLES :
C              SBJECT  name of feature being modified.
C              NTAB    no. of decision table being modified.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  DCV = designer-controlled variable.
C*************************************************************************
      CHARACTER SBJECT*16,REPLY*1,STRING(20),UNITS*10,TABLE,EQUATN*70
      CHARACTER*20 LKAXIS(15,5),LKNAME(15),NAME*20
      CHARACTER*30 TITLE(100),CASE(6),MESAGE(100)
      INTEGER NDEPN(50),NCOORD(5),NOLDEQ(80),NEWEQN(80),NSORCE(50)
C
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
      COMMON/BLOK8/NDCV(100,30)
C
      NWFLAG=0
C
C     * check for new variables etc arising from edit *
   70 CALL DESPGM
      WRITE(*,101)
      READ(*,201)NANSWR
      GOTO(71,72,73,74)NANSWR
      GOTO 70
C
C     ** modify a decision table,  display list of existing tables **
   71 WRITE(*,102)
      DO 41 KTAB=1,NDETAB
C        * check whether table is active *
```

```
         IF(NCOND(KTAB).EQ.0)GOTO 41
         WRITE(*,110)KTAB,TITLE(KTAB)
   41 CONTINUE
      WRITE(*,103)
      READ(*,202)NTAB
      IF(NANSWR.GT.NDETAB)GOTO 71
C     * show nominated table *
      CALL SHOW(NTAB)
C     * what aspect of table is to be modifed? *
  710 WRITE(*,104)
      READ(*,201)NANSWR
      GOTO(711,712,713,714,70,74)NANSWR
      GOTO 710
C
C     * delete table *
  711 WRITE(*,105)TITLE(NTAB)
      READ(*,203)REPLY
      IF(REPLY.EQ.'Y'.OR.REPLY.EQ.'y')THEN
C        * delete table *
         NCOND(NTAB)=0
         NACT(NTAB)=0
         NRULE(NTAB)=0
         NDCV(NTAB,1)=0
         TITLE(KTAB)=' '
C        * (possible development - add no. to a list of vacant tables) *
      END IF
      GOTO 70
C     * delete table ends *
C
C     * modify condition stub *
  712 SBJECT='a condition'
      WRITE(*,106)SBJECT,SBJECT,SBJECT
      READ(*,201)NANSWR
      GOTO(7121,7122,7123,70)NANSWR
      GOTO 712
C        * delete a condition *
 7121    CALL SHOW(NTAB)
         WRITE(*,107)
         READ(*,202)NANSWR
         NROW=NCOND(NTAB)+NACT(NTAB)
         DO 42 KROW=NANSWR,(NROW-1)
            DO 421 KRULE=1,NRULE(NTAB)
               TABLE(NTAB,KROW,KRULE)=TABLE(NTAB,KROW+1,KRULE)
  421       CONTINUE
            NSTORE(NTAB,KROW)=NSTORE(NTAB,KROW+1)
   42    CONTINUE
C        * reduce number of rows *
         NCOND(NTAB)=NCOND(NTAB)-1
         CALL SHOW(NTAB)
         GOTO 70
C        * append a condition *
C        * make room for new condition *
 7122    NROW=NCOND(NTAB)+NACT(NTAB)+1
         DO 43 KROW=1,NACT(NTAB)
```

```
            JROW=NROW+1-KROW
            DO 431 KRULE=1,NRULE(NTAB)
               TABLE(NTAB,JROW,KRULE)=TABLE(NTAB,JROW-1,KRULE)
  431       CONTINUE
            NSTORE(NTAB,JROW)=NSTORE(NTAB,JROW-1)
   43     CONTINUE
          NCOND(NTAB)=NCOND(NTAB)+1
C         * get new condition *
          NEDIT=1
          CALL DETAB(NTAB,NEDIT,NCOND(NTAB))
          CALL SHOW(NTAB)
          GOTO 70
C         * re-enter a condition *
 7123     CALL SHOW(NTAB)
          WRITE(*,107)
          READ(*,202)NANSWR
          NEDIT=1
          CALL DETAB(NTAB,NEDIT,NANSWR)
          CALL SHOW(NTAB)
          GOTO 70
C     * modify condition stub ends *
C
C     * modify action stub *
  713 SBJECT='an action'
      WRITE(*,106)SBJECT,SBJECT,SBJECT
      READ(*,201)NANSWR
      GOTO(7131,7132,7133,70)NANSWR
      GOTO 713
C         * delete an action *
 7131     CALL SHOW(NTAB)
          WRITE(*,108)
          READ(*,202)NANSWR
          NANSWR=NANSWR+NCOND(NTAB)
          NROW=NCOND(NTAB)+NACT(NTAB)
          DO 44 KROW=NANSWR,(NROW-1)
             KACT=KROW-NCOND(NTAB)
             DO 441 KRULE=1,NRULE(NTAB)
                TABLE(NTAB,KROW,KRULE)=TABLE(NTAB,KROW+1,KRULE)
  441        CONTINUE
             NSTORE(NTAB,KROW)=NSTORE(NTAB,KROW+1)
             NTYPE(NTAB,KACT)=NTYPE(NTAB,KACT+1)
             METHOD(NTAB,KACT)=METHOD(NTAB,KACT+1)
   44     CONTINUE
C         * reduce number of rows *
          NACT(NTAB)=NACT(NTAB)-1
          CALL SHOW(NTAB)
          GOTO 70
C         * append an action *
 7132     NACT(NTAB)=NACT(NTAB)+1
          NEDIT=2
          CALL DETAB(NTAB,NEDIT,NACT(NTAB))
          GOTO 70
C         * re-enter an action *
 7133     CALL SHOW(NTAB)
```

```
         WRITE(*,108)
         READ(*,202)NANSWR
         NEDIT=2
         CALL DETAB(NTAB,NEDIT,NANSWR)
         CALL SHOW(NTAB)
         GOTO 70
C     * modify action stub ends *
C
C     * modify rule *
  714 SBJECT='a rule'
      WRITE(*,106)SBJECT,SBJECT,SBJECT
      READ(*,201)NANSWR
      GOTO(7141,7142,7143,70)NANSWR
      GOTO 714
C        * delete a rule *
 7141    CALL SHOW(NTAB)
         WRITE(*,109)
         READ(*,202)NANSWR
         NROW=NCOND(NTAB)+NACT(NTAB)
         DO 45 KRULE=NANSWR,(NRULE(NTAB)-1)
            DO 45 KROW=1,NROW
               TABLE(NTAB,KROW,KRULE)=TABLE(NTAB,KROW,KRULE+1)
   45    CONTINUE
C        * reduce number of rules *
         NRULE(NTAB)=NRULE(NTAB)-1
         CALL SHOW(NTAB)
         GOTO 70
C        * append an rule *
 7142    NRULE(NTAB)=NRULE(NTAB)+1
         NEDIT=3
         CALL DETAB(NTAB,NEDIT,NRULE(NTAB))
         CALL SHOW(NTAB)
         GOTO 70
C        * re-enter a rule *
 7143    CALL SHOW(NTAB)
         WRITE(*,109)
         READ(*,202)NANSWR
         NEDIT=3
         CALL DETAB(NTAB,NEDIT,NANSWR)
         CALL SHOW(NTAB)
         GOTO 70
C     * modify rule ends *
C
C     ** table modification ends **
C
C     ** modify a variable **
   72 CALL GETVAR(NADRES)
      NPREVS=0
      IF(NADRES.NE.0)THEN
C        * variable known *
         READ(12,205,REC=NADRES)NEQN,NTABD,UNITS,NAME,
     +      (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
  721    WRITE(*,113)
         READ(*,201)NANSWR
```

```
          GOTO(722,723,70,74)NANSWR
          GOTO 721
C         * modify units *
  722     WRITE(*,114)UNITS
          READ(*,203)UNITS
C          * write data back to file *
          WRITE(12,205,REC=NADRES)NEQN,NTABD,UNITS,NAME,
     +       (NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
          GOTO 70
C         * modify equation *
  723     IF(NEQN.GT.0)THEN
C            * list all existing equations for this variable *
             WRITE(*,125)NAME
  724        NPREVS=NEQN
             READ(15,206,REC=NPREVS)NEQN,EQUATN
             IF(EQUATN.NE.'      ')THEN
                WRITE(*,112)NPREVS,EQUATN
             ENDIF
             IF(NEQN.NE.0)THEN
                GOTO 724
             END IF
             WRITE(*,115)
             READ(*,202)NEQN
             READ(15,208,REC=NEQN)NOLDEQ(1),
     +                                (NOLDEQ(J),J=2,(NOLDEQ(1)+1))
          ELSE
C            * variable is designer-controlled *
             NUMEQN=NUMEQN+1
             NEQN=NUMEQN
          END IF
          CALL INPUT(NAME)
          NPREVS=0
          LENGTH=70
          CALL ENCODE(NPREVS,NEQN,MIDARG,LENGTH,NADRES)
C         * read in new rev. polish eqn *
          READ(15,208,REC=NEQN)NEWEQN(1),(NEWEQN(J),J=2,(NEWEQN(1)+1))
C         * check for changes in ingredients list (ie equation) *
C         * take each var in old eqn and see whether in new eqn *
          DO 46 KOLD=2,(NOLDEQ(1)+1)
             IF(NOLDEQ(KOLD).GT.50)THEN
C               * else eqn element is an operator or look-up table *
                DO 47 KNEW=2,(NEWEQN(1)+1)
                   IF(NOLDEQ(KOLD).EQ.NEWEQN(KNEW))THEN
C                     * element is in new eqn *
                      GOTO 46
                   ENDIF
  47            CONTINUE
C               * normal exit from loop 47 implies var not in new eqn *
C               * remove var from NOLDEQ(KOLD)'s dependents list *
                CALL UNDEPN(NOLDEQ(KOLD),NADRES)
             ENDIF
  46      CONTINUE
C       ELSE
C          * variable unknown - name incorrectly entered *
```

```
      ENDIF
      GOTO 70
C     ** variable modification ends **
C
C     ** modify a look up table **
C     * give options *
   73 WRITE(*,118)
      READ(*,201)NREPLY
      GOTO(731,731,731,70,74)NREPLY
      GOTO 73
C     * give list of active look-up tables *
  731 CALL GETLK(LKNAME,LKDIM,NLKUP,LKNO)
      GOTO(732,733,734)NREPLY
      GOTO 73
C     * single entry replacement, identify location *
  732 CALL GETXYZ(LKNO,LKAXIS,NCOORD,LKDIM)
C     * identify data location in RLKDATA array *
      CALL LKPOSN(LKNO,LKDIM,LKSIZE,LKPTR,NCOORD,LOCATE)
      WRITE(*,122)RLKDAT(LOCATE)
      READ(*,207)RLKDAT(LOCATE)
      GOTO 70
C
C     * re-enter an entire table *
  733 WRITE(*,123)
      READ(*,203)REPLY
      IF(REPLY.EQ.'Y'.OR.REPLY.EQ.'y')THEN
         CALL LKLOAD(LKNO)
      ENDIF
      GOTO 70
C
C     * delete a look-up table, query and confirm first *
  734 WRITE(*,124)LKNAME(LKNO)
      READ(*,203)REPLY
      IF(REPLY.EQ.'y'.OR.REPLY.EQ.'Y')THEN
         LKDIM(LKNO)=0
         LKPTR(LKNO)=0
      ENDIF
      GOTO 70
C
C     ** lookup table modification ends **
C
   74 RETURN
C
  101 FORMAT(//' MASTER MODIFICATION MENU'
     +       /' ----------------------'/
     +         ' Do you wish to alter a -'
     +   //1x,'1  decision table'
     +    /1x,'2  variable (units or equation)'
     +    /1x,'3  look up table'
     +    /1x,'4  quit modification, return to main menu'
     + /1x/1x,':: enter number of choice - '$)
  102 FORMAT(' :: the following are existing or reserved tables :-'
     +/1X,/4X,'TABLE NUMBER',8X,'TITLE'/1X)
  103 FORMAT(/1X,/1X,':: enter number of table - '$)
```

```
104 FORMAT(1x/1x/1x,' Which aspect of the table is to be modified?'
   +/1x/1x,'1  entire table (delete)'
   +   /1x,'2  condition    (delete, append, re-enter)'
   +   /1x,'3  action       (delete, append, re-enter)'
   +   /1x,'4  rule         (delete, append, re-enter)'
   +   /1x,'5  return to master modification menu'
   +   /1x,'6  quit modification, return to main menu'
   +/1x/1x,':: enter number of choice - '$)
105 FORMAT(/1X,/' Delete table ',A
   +//' :: ARE YOU SURE !! (Y/N) - '$)
106 FORMAT(/1X,/1X,' Do you wish to  -'
   +/1x,/1x,'1 delete ',A
   +    /1X,'2 append ',A
   +    /1x,'3 re-enter ',A
   +    /1X,'4 quit'/1x/1x,':: enter number of choice - '$)
107 FORMAT(/1X,/1X,':: enter condition number - '$)
108 FORMAT(/1X,/1X,':: enter action number - '$)
109 FORMAT(/1X,/1X,':: enter rule number - '$)
110 FORMAT(10X,I2,7X,A)
111 FORMAT(' :: enter name of variable - '$)
112 FORMAT(4X,I4,3X,A)
113 FORMAT(///' Which aspect of the variable is to be modified?'
   +/1x/1x,'1  units'
   +   /1x,'2  an equation'
   +   /1x,'3  return to master modification menu'
   +   /1x,'4  quit modification, return to main menu'
   +/1x/1x,':: enter number of choice - '$)
114 FORMAT(/1X,/1X,'Current units are ',A/1x,':: enter new units - '$)
115 FORMAT(/1X,/1X,':: enter number of equation to be re-entered - '$)
118 FORMAT(/1X/' Do you wish to -'
   +       /1x/' 1  re-enter a single table entry'
   +          /' 2  re-enter an entire look-up table'
   +          /' 3  delete a look-up table'
   +          /' 4  return to master modification menu'
   +          /' 5  quit modification, return to main menu'
   +       /1x/' :: enter number of choice - '$)
122 FORMAT(/1X/' Current value is ',3P1E12.2E2
   + /' :: enter new value (up to 12 digits including a compulsory'
   + /'    decimal point and optional 2 digit exponent) '$)
123 FORMAT(/1X/' This will allow changes in table size but not in the'
   +   /' number of axes. If this is required edit the equation '
   +   /' concerned to use a new look-up table name and delete the '
   +   /' original table via the modify option on the main menu.'
   +   /1x/' :: do you wish to continue?  (Y/N) - '$)
124 FORMAT(/1X/' :: ARE YOU SURE you want to DELETE look-up table ',A,
   +   '  (Y/N) - ' $)
125 FORMAT(/1X/' The following are all the equations and
   +equation '/' numbers for the calculation of ',A,/1X)
201 FORMAT(I1)
202 FORMAT(I2)
203 FORMAT(A)
204 FORMAT(20A1)
```

```
205 FORMAT(I4,1X,I4,1X,A10,1X,A20,50I4)
206 FORMAT(I2,A)
207 FORMAT(E12.2E2)
208 FORMAT(82X,80I4)
    END
```

```
      SUBROUTINE RPN(MIDEQN,NEQN)
C*************************************************************************
C   PURPOSE :  To convert an equation encoded into numeric form (by
C              ENCODE) to RPN.
C   METHODS :  A two pointer system is used to identify operations,
C              functions, etc. & decide which of the two takes
C              precedence. If the leading pointer has priority there may
C              be a third element which takes precedence over both. At
C              this point stacking occurs: a marker is inserted at the
C              position of the leading pointer which is then incremented
C              in search of further elements. The stacking and insertion
C              of extra markers may occur several times before
C              operations returns to the original markers.
C   ROUTINES CALLED :
C              EQN      enters elements into the Rev. Polish eqn.
C   ARGUMENTS :
C              MIDEQN   array holding the encoded form of the original
C                       equation from ENCODE.
C              NEQN     no. of eqn being processed.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C              LKPOSN1 markers pointing to locations in MIDEQN.
C              LSTACK  records type of eqn entry where markers inserted.
C   LOCAL VARIABLES :
C              IPOSN2  position of leading pointer in MIDEQN.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  Double logicals are not permitted.
C*************************************************************************
      CHARACTER*82 HOLD
      INTEGER LPOSN1(30),LSTACK(30),MIDEQN(30)
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK9/NRPNEQ(50)
C
      LEVEL=0
      NFLAG=0
      NEND=0
      IPOSN2=1
      LPOSN1(1)=0
      NRPNEQ(1)=0
      LSTACK(1)=1
C
C     * stacking segment *
   70 LEVEL=LEVEL+1
C     * stack ends *
C
C     * step segment *
   71 IPOSN2=IPOSN2+1
      IF(NEND.EQ.0.AND.MIDEQN(IPOSN2).GT.50)THEN
C        * var addres at IPOSN2 pointer, put into RPN eqn *
         CALL EQN(MIDEQN(IPOSN2))
      ENDIF
      IF(IPOSN2.GE.(MIDEQN(1)+1).OR.MIDEQN(IPOSN2).EQ.11)THEN
C        * end of eqn or right bracket *
```

```
         IF(IPOSN2.GE.(MIDEQN(1)+1))NEND=1
         NARG=LPOSN1(LEVEL)
         IF(LPOSN1(LEVEL).NE.0)CALL EQN(MIDEQN(NARG))
         LPOSN1(LEVEL)=0
         GOTO 73
       END IF
C      * step ends *
C
   72 IF(MIDEQN(IPOSN2).LE.9)THEN
C        * operator detected, skip any commas in input eqn *
         IF(MIDEQN(IPOSN2).EQ.4)GOTO 71
         IF(LPOSN1(LEVEL).GT.0)THEN
C           * no.1 flag is active, see which element takes precedence *
            IF(MIDEQN(IPOSN2).GT.MIDEQN(LPOSN1(LEVEL)))THEN
C              * stack *
               LPOSN1(LEVEL+1)=IPOSN2
               LSTACK(LEVEL)=1
               GO TO 70
            ELSE
C              * 1st operation has precedence and may be placed in eqn *
               NARG=LPOSN1(LEVEL)
               CALL EQN(MIDEQN(NARG))
               IF(LEVEL.GT.1.AND.LSTACK(LEVEL-1).EQ.1)THEN
                  LPOSN1(LEVEL)=0
                  LEVEL=LEVEL-1
                  GOTO 72
               END IF
               LPOSN1(LEVEL)=IPOSN2
               GOTO 73
            ENDIF
C
C           * because of different codes - + (or * /) at pointers 1&2
C           * registers  different precedent to + - (or / *) at
C           * pointers 1&2 respectively, hence the following check *
            IF(MIDEQN(LPOSN1(LEVEL)).EQ.5.AND.MIDEQN(IPOSN2).EQ.6.OR.
     +         MIDEQN(LPOSN1(LEVEL)).EQ.7.AND.MIDEQN(IPOSN2).EQ.8)THEN
C              * operators have equal precedence, put 1st op into eqn *
C              * note: opposite case (+ - or / *) taken care of by
C              *       ordinary evaluation of relative precedents *
               NARG=LPOSN1(LEVEL)
               CALL EQN(MIDEQN(NARG))
               IF(LEVEL.GT.1.AND.LSTACK(LEVEL-1).EQ.1)THEN
                  LPOSN1(LEVEL)=0
                  LEVEL=LEVEL-1
                  GOTO 72
               END IF
               LPOSN1(LEVEL)=IPOSN2
               GOTO 73
            ENDIF
C
         ELSE
C           * no.1 flag is not active - use it as a marker and
C           * look for higher precedence operations *
            LPOSN1(LEVEL)=IPOSN2
```

```
            GOTO 71
         END IF
      END IF
      IF(MIDEQN(IPOSN2).EQ.10)THEN
C        * don't want ( copied *
         IF(LPOSN1(LEVEL).EQ.0.OR.MIDEQN(LPOSN1(LEVEL)).LE.9)THEN
C           * ( follows an operation or is at beginning of eqn *
            LSTACK(LEVEL)=2
         ELSE
C           * ( follows a function or look-up table name *
            LSTACK(LEVEL)=3
         END IF
         LPOSN1(LEVEL+1)=0
         GOTO 70
      END IF
      IF(MIDEQN(IPOSN2).LE.50)THEN
C        * lookup table name or function *
         LPOSN1(LEVEL+1)=IPOSN2
         LSTACK(LEVEL)=4
         GOTO 70
      END IF
      GOTO 71
   73 IF(LPOSN1(LEVEL).NE.0)GOTO 71
C
C     * unstack *
   74 IF(LEVEL.EQ.1)THEN
         IF(NEND.NE.1)GOTO 71
         READ(15,11,REC=NEQN)HOLD
         WRITE(15,10,REC=NEQN)HOLD,(NRPNEQ(J),J=1,NRPNEQ(1)+1)
C
         RETURN
C
      ELSE
         LEVEL=LEVEL-1
         GOTO(71,71,74,75)LSTACK(LEVEL)
   75    NARG=LPOSN1(LEVEL+1)
         CALL EQN(MIDEQN(NARG))
         GOTO 73
      END IF
C     * end of unstacking segment *
      STOP
   10 FORMAT(A82,80I4)
   11 FORMAT(A82)
      END
```

```
      SUBROUTINE SHOW(NTAB)
C*************************************************************************
C   PURPOSE :      To display a decision table.
C   ARGUMENTS :    NTAB  no. of decision table to be displayed.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS : STUB  holds condition and action stub contents.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*************************************************************************
      CHARACTER STUB(20),TABLE,REPLY*1
      CHARACTER*30 TITLE(100),MESAGE(100)
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK5/TITLE
C
      WRITE(*,101)TITLE(NTAB)
C     * display conditions *
      DO 43 KCOND=1,NCOND(NTAB)
         NREC=NSTORE(NTAB,KCOND)
         READ(12,204,REC=NREC)(STUB(J),J=1,20)
         WRITE(*,106)(STUB(J),J=1,20),
     +      (TABLE(NTAB,KCOND,J),J=1,NRULE(NTAB))
   43 CONTINUE
C     * separate conditions and actions *
      WRITE(*,103)
      DO 44 KRULE=1,NRULE(NTAB)
         WRITE(*,107)
   44 CONTINUE
      WRITE(*,108)
C     * display actions *
      DO 45 KACT=1,NACT(NTAB)
         KROW=NCOND(NTAB)+KACT
         IF(NTYPE(NTAB,KACT).EQ.1)THEN
C           * calc. value of a variable *
            NREC=NSTORE(NTAB,KROW)
            READ(12,204,REC=NREC)(STUB(J),J=1,20)
            WRITE(*,109)(STUB(J),J=1,20),
     +         (TABLE(NTAB,KROW,J),J=1,NRULE(NTAB))
         ELSEIF(NTYPE(NTAB,KACT).EQ.2)THEN
C           * display value of a variable *
            NREC=NSTORE(NTAB,KROW)
            READ(12,204,REC=NREC)(STUB(J),J=1,20)
            WRITE(*,110)(STUB(J),J=1,20),
     +         (TABLE(NTAB,KROW,J),J=1,NRULE(NTAB))
         ELSEIF(NTYPE(NTAB,KACT).EQ.3)THEN
C        * execute a decision table *
            WRITE(*,111)TITLE(METHOD(NTAB,KACT)),
     +         (TABLE(NTAB,KROW,J),J=1,NRULE(NTAB))
         ELSEIF(NTYPE(NTAB,KACT).EQ.4)THEN
C           * display a message *
            WRITE(*,112)MESAGE(METHOD(NTAB,KACT)),
     +         (TABLE(NTAB,KROW,J),J=1,NRULE(NTAB))
         ENDIF
   45 CONTINUE
```

```
      WRITE(*,102)
      READ(*,201)REPLY
      RETURN
C
  101 FORMAT(//' decision table ',A/)
  102 FORMAT(//' press <RETURN> to continue ....'$)
  103 FORMAT(1X)
  106 FORMAT(1X,'     ',20A1,'? || ',50A1)
  107 FORMAT('+='$)
  108 FORMAT('+',30('='))
  109 FORMAT(1X,'calc ',20A1,'  || ',50A1)
  110 FORMAT(1X,'show ',20A1,'  || ',50A1)
  111 FORMAT(1X,'do   ',A20,'  || ',50A1)
  112 FORMAT(1X,'say  ',A20,'  || ',50A1)
  201 FORMAT(A)
  204 FORMAT(21X,20A1)
      END
```

```
      SUBROUTINE TABEX3(NTNUM,NCASE,NERROR)
C*****************************************************************
C   PURPOSE :  Decision table execution and variable evaluation.
C   METHODS :  Decision table execution uses 2 nested loops to scan
C              rows and rules. DCV's are reviewed before execution
C              begins. Variable evaluation uses a push-down
C              stack for calculations.
C              Principle of recursive descent is used; stacking &
C              unstacking routines enable execution to be suspended at a
C              particular level, then resumed once missing data has been
C              evaluated.
C   ROUTINES CALLED :  WARN3
C                      LKPOSN
C   ARGUMENTS :  NTNUM  number of first decision table to be executed.
C                NCASE  redundant.
C                NERROR error flag to signal calling program.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C                    MISING  address of undefined variable being
C                            evaluated at each level of operations.
C                    NTABLE  no. of decision table being
C                            executed at each level of operations.
C                    LEQN    equation used to evaluate MISING.
C                    LCALC   pointer to current position in LEQN.
C                    LEQUAT  no. of eqn in use at each level of ops.
C                    NCOORD  look-up table parameters.
C                    LSTKCK  check array for calculation stack height.
C                    LDCVDT  decision table no. for new dcvs encountered.
C                    STACK   calculation stack.
C   LOCAL VARIABLES :
C                    LSTACK  current calculation stack level.
C                    NSFLAG  =1 if stacking ocurred to reach this level.
C                    NVFLAG  =1 if a dcv is to be removed from dcv list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  Output from TABEX3 goes to screen and OUTPUT.dat.
C              DCV = designer-controlled variable.
C              Trig. functions use radians.
C              'Stacking' and 'level of operations' refer to recursive
C              descent processing of decision tables and data.
C              Calculation stack refers to array used to hold values of
C              data during calculation.
C              TABEX3 writes to TRACE.dat to assist with debugging.
C*****************************************************************
      CHARACTER UNITS*10,NAME*20,CSTUB*1,TABLE,DATA*12
      CHARACTER*20 LKAXIS(15,5),LKNAME(15)
      CHARACTER*30 TITLE(100),CASE(6),MESAGE(100)
      INTEGER MISING(15),LRULE(15),LCOND(15),LACT(15),NTABLE(15),
     +        LEQN(15,80),LCALC(15),LEQUAT(15),NCOORD(5),LSTKCK(15),
     +        LDCVDT(15)
      REAL STACK(100)
C
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
```

```
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
      COMMON/BLOK5/TITLE
      COMMON/BLOK6/LKDIM(15),LKSIZE(15,5),LKPTR(15),
     +   LKTOT(15),RLKDAT(100),LKNEXT
      COMMON/BLOK6A/LKAXIS,LKNAME,CASE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
      COMMON/BLOK8/NDCV(100,30)
C
      LSTACK=0
      LEVEL=0
      NERROR=0
      NWFLAG=0
      NTABLE(1)=NTNUM
      LDCVDT(1)=NTNUM
C
      WRITE(10,9901)NTNUM
C
C     * stacking *
   71 LEVEL=LEVEL+1
C     * initialise arrays for this level *
      LCALC(LEVEL)=2
      LRULE(LEVEL)=1
      LCOND(LEVEL)=1
      LACT(LEVEL)=1
      NSFLAG=1
      WRITE(10,9991)LEVEL
C     * end of stacking segment *
C
   77 NTNUM=NTABLE(LEVEL)
C     * is evaluation of a table or ingredient required? *
      IF(NTNUM.GE.1)THEN
      WRITE(10,9992)NTNUM
C
C        * table execution *
C
         IF(NSFLAG.EQ.1)THEN
C           * review DCV's on list associated with this table *
C           * (watch for DCV's not yet on a list during execution) *
C           * if NPRD=1 for a DCV then value has been preset and is
C           * not to be queried each run *
            NVFLAG=0
            DO 401 KDCV=2,(NDCV(NTNUM,1)+1)
               NARG=NDCV(NTNUM,KDCV)
               IF(NPRD(1,NARG).EQ.0)THEN
C                 * DCV has been neither been preset nor queried by
C                 * another decision table this run - review value *
                  READ(12,204,REC=NARG)NEQN,NTABD,UNITS,NAME,NDEP
C                 * if NEQN or NTABD>0 var. is no longer a dcv,
C                 * remove from list *
                  IF(NEQN.NE.0.OR.NTABD.NE.0)THEN
C                    * mark dcv for removal from list *
                     NDCV(NTNUM,KDCV)=0
```

```
                  NVFLAG=1
                  GOTO 401
               ENDIF
C
               IF(NDEP.EQ.0)THEN
C                 * dcv is a condition stub *
  909              IF(RCASE(1,NARG).EQ.0.0)THEN
                     CSTUB='N'
                  ELSEIF(RCASE(1,NARG).EQ.1.0)THEN
                     CSTUB='Y'
                  ENDIF
                  WRITE(*,1071)NAME,CSTUB
                  WRITE(10,1071)NAME,CSTUB
                  READ(*,205)DATA
                  IF(DATA.NE.'            ')THEN
C                    * designer has altered value *
                     IF(DATA.EQ.'Y'.OR.DATA.EQ.'y')THEN
                        RCASE(1,NARG)=1.0
                     ELSEIF(DATA.EQ.'N'.OR.DATA.EQ.'n')THEN
                        RCASE(1,NARG)=0.0
                     ELSE
C                       * invalid entry - try again *
                        GOTO 909
                     ENDIF
                  ENDIF
               ELSE
C                 * dcv is an ordinary variable *
                  WRITE(*,107)NAME,RCASE(1,NARG),UNITS
                  WRITE(10,107)NAME,RCASE(1,NARG),UNITS
                  READ(*,205)DATA
                  IF(DATA.NE.'            ')THEN
C                    * designer has altered data value *
                     WRITE(12,205,REC=2)DATA
                     READ(12,202,REC=2)RCASE(1,NARG)
C                    * set data presence flags of
C                    * dependant data to 0 *
                     CALL WARN3(NARG,NWFLAG)
                  ENDIF
               ENDIF
               NPRD(1,NARG)=1
C              * record address for reseting at end of run *
               NRESET(1)=NRESET(1)+1
               NRESET(NRESET(1)+1)=NARG
C           ELSE
C              * value already set *
            ENDIF
  401     CONTINUE
C
C         * check if dcv list is to be updated *
          IF(NVFLAG.EQ.1)THEN
C            * at least one dcv to be removed from list *
             NUMDCV=NDCV(NTNUM,1)+1
             DO 402 K402=2,NUMDCV
                IF(NDCV(NTNUM,K402).EQ.0)THEN
```

```
C                    * remove list entry *
                     DO 403 K403=K402,NDCV(NTNUM,1)
                        NDCV(NTNUM,K403)=NDCV(NTNUM,K403+1)
  403                CONTINUE
                     NDCV(NTNUM,1)=NDCV(NTNUM,1)-1
                  ENDIF
  402             CONTINUE
               ENDIF
C
            ENDIF
C           * initialise loop control variables *
            LR=LRULE(LEVEL)
            NREND=NRULE(NTNUM)
            LC=LCOND(LEVEL)
            NCEND=NCOND(NTNUM)
C           * outer loop sets rule number, inner sets condition number *
            DO 41 KRULE=LR,NREND
C              * after unstacking don't want to re-check conditions checked
C              * before execution suspended *
               IF(KRULE.EQ.LRULE(LEVEL))THEN
                  LC=LCOND(LEVEL)
               ELSE
                  LC=1
               ENDIF
               DO 42 KCOND=LC,NCEND
C                 * is condition entry immaterial? *
                  IF(TABLE(NTNUM,KCOND,KRULE).EQ.'-')GOTO 42
C                 * find condition adress and check whether data present *
                  NADRES=NSTORE(NTNUM,KCOND)
                  IF(NPRD(1,NADRES).EQ.0)THEN
C                    * data not present - prepare to stack *
                     LRULE(LEVEL)=KRULE
                     LCOND(LEVEL)=KCOND
                     READ(12,106,REC=NADRES)NEQN,NTABD
                     NTABLE(LEVEL+1)=NTABD
                     MISING(LEVEL+1)=NADRES
                     LEQUAT(LEVEL+1)=NEQN
                     LDCVDT(LEVEL+1)=NTNUM
                     GOTO 71
                  ENDIF
C                 * match condition entry with corresponding condition in
C                 * condition stub. If no match go to next rule *
C                 * convert real value stored in RCASE to logical *
                  N=AINT(RCASE(1,NADRES))
                  IF(N.EQ.1)THEN
                     CSTUB='Y'
                  ELSE IF(N.EQ.0)THEN
                           CSTUB='N'
                  ELSE
C                    * error - non logical entry *
                     WRITE(*,95)NTNUM,KCOND,N
                     RETURN
                  ENDIF
                  IF(CSTUB.NE.TABLE(NTNUM,KCOND,KRULE))GOTO 41
```

```
   42       CONTINUE
C           * normal exit from loop 42 implies appropriate rule
C           * has been found *
            LASTRL(1,NTNUM)=KRULE
C
C           * action processing *
            LA=LACT(LEVEL)
            NACEND=NACT(NTNUM)
C           * store essential information in case stacking require *
            LRULE(LEVEL)=KRULE
            LCOND(LEVEL)=NCOND(NTNUM)
            DO 44 KACT=LA,NACEND
               KROW=KACT+NCOND(NTNUM)
               IF(TABLE(NTNUM,KROW,KRULE).EQ.'X')THEN
C                 * carry out action *
                  LACT(LEVEL)=KACT+1
                  IF(NTYPE(NTNUM,KACT).EQ.1)THEN
C                    * calculate a variable *
                     WRITE(*,110)TITLE(NTNUM),NTNUM,KRULE
                     WRITE(10,110)TITLE(NTNUM),NTNUM,KRULE
                     WRITE(11,110)TITLE(NTNUM),NTNUM,KRULE
                     NTABLE(LEVEL+1)=0
                     MISING(LEVEL+1)=NSTORE(NTNUM,KROW)
                     LEQUAT(LEVEL+1)=METHOD(NTNUM,KACT)
                     LDCVDT(LEVEL+1)=NTNUM
                     GOTO 71
                  ELSEIF(NTYPE(NTNUM,KACT).EQ.2)THEN
C                    * display the value of a variable - get name *
                     READ(12,201,REC=NSTORE(NTNUM,KROW))UNITS,NAME
                     IF(NPRD(1,NSTORE(NTNUM,KROW)).EQ.0)THEN
C                       * var undefined *
                        WRITE(*,108)TITLE(NTNUM),NTNUM,KRULE,NAME
                        WRITE(10,108)TITLE(NTNUM),NTNUM,KRULE,NAME
                        WRITE(11,108)TITLE(NTNUM),NTNUM,KRULE,NAME
                     ELSE
                        WRITE(*,109)TITLE(NTNUM),NTNUM,KRULE,NAME,
     +                     RCASE(1,NSTORE(NTNUM,KROW)),UNITS
                        WRITE(10,109)TITLE(NTNUM),NTNUM,KRULE,NAME,
     +                     RCASE(1,NSTORE(NTNUM,KROW)),UNITS
                        WRITE(11,109)TITLE(NTNUM),NTNUM,KRULE,NAME,
     +                     RCASE(1,NSTORE(NTNUM,KROW)),UNITS
                     ENDIF
                  ELSE IF(NTYPE(NTNUM,KACT).EQ.3)THEN
C                    * execute a decision table *
                     WRITE(*,110)TITLE(NTNUM),NTNUM,KRULE
                     WRITE(10,110)TITLE(NTNUM),NTNUM,KRULE
                     WRITE(11,110)TITLE(NTNUM),NTNUM,KRULE
                     NTABLE(LEVEL+1)=METHOD(NTNUM,KACT)
                     LDCVDT(LEVEL+1)=NTABLE(LEVEL+1)
                     GOTO 71
                  ELSE IF(NTYPE(NTNUM,KACT).EQ.4)THEN
C                    * display a message *
                     WRITE(*,110)TITLE(NTNUM),NTNUM,KRULE,
     +                  MESAGE(METHOD(NTNUM,KACT))
```

```
                  WRITE(10,110)TITLE(NTNUM),NTNUM,KRULE,
     +               MESAGE(METHOD(NTNUM,KACT))
                  WRITE(11,110)TITLE(NTNUM),NTNUM,KRULE,
     +               MESAGE(METHOD(NTNUM,KACT))
               ENDIF
            ENDIF
   44       CONTINUE
            GOTO 72
C           * end of action processing *
C
   41    CONTINUE
C        * normal exit from loop 41 implies no matching rule found *
         NERROR=1
         WRITE(*,91)TITLE(NTNUM),NTNUM
         WRITE(10,91)TITLE(NTNUM),NTNUM
         WRITE(11,91)TITLE(NTNUM),NTNUM
         RETURN
C
      ELSE
C
C        * a data element is to be evaluated *
         NADRES=MISING(LEVEL)
         WRITE(10,9994)NADRES,LSTACK,LSTKCK(LEVEL)
C        * is variable designer-controlled? *
   79    IF(LEQUAT(LEVEL).EQ.0)THEN
C           * variable is a DCV not on the DCV list
C           * (otherwise would have been queried before exec. began) *
            NARG=NADRES
            READ(12,204,REC=NARG)NEQN,NTABD,UNITS,NAME,NDEP
            IF(NDEP.EQ.0)THEN
C              * dcv is a condition stub *
  910          IF(RCASE(1,NARG).EQ.0.0)THEN
                  CSTUB='N'
                  ELSEIF(RCASE(1,NARG).EQ.1.0)THEN
                     CSTUB='Y'
               ENDIF
               WRITE(*,1071)NAME,CSTUB
               WRITE(10,1071)NAME,CSTUB
               READ(*,205)DATA
               IF(DATA.NE.'            ')THEN
C                 * designer has altered value *
                  IF(DATA.EQ.'Y'.OR.DATA.EQ.'y')THEN
                     RCASE(1,NARG)=1.0
                  ELSEIF(DATA.EQ.'N'.OR.DATA.EQ.'n')THEN
                     RCASE(1,NARG)=0.
                  ELSE
C                    * invalid entry - try again *
                     GOTO 910
                  ENDIF
               ENDIF
            ELSE
C              * dcv is an ordinary variable *
               WRITE(*,107)NAME,RCASE(1,NARG),UNITS
               WRITE(10,107)NAME,RCASE(1,NARG),UNITS
```

```
                READ(*,205)DATA
                IF(DATA.NE.'              ')THEN
C                  * designer has altered data value *
                   WRITE(12,205,REC=2)DATA
                   READ(12,202,REC=2)RCASE(1,NARG)
C                  * set data presence flags of
C                  * dependant data to 0 *
                   CALL WARN3(NARG,NWFLAG)
                ENDIF
             ENDIF
             NPRD(1,NARG)=1
C            * record address for reseting at end of run *
             NRESET(1)=NRESET(1)+1
             NRESET(NRESET(1)+1)=NARG
C            * put variable onto DCV list *
             NDT=LDCVDT(LEVEL)
             NDCV(NDT,1)=NDCV(NDT,1)+1
             NDCV(NDT,(NDCV(NDT,1)+1))=NADRES
          ELSE
C            * not a DCV - calculation required *
C            * if stack height increasing read in eqn for this level *
             IF(NSFLAG.EQ.1)THEN
                READ(15,203,REC=LEQUAT(LEVEL))LEQN(LEVEL,1),
     +             (LEQN(LEVEL,J),J=2,(LEQN(LEVEL,1)+1))
                IF(LEQN(LEVEL,1).EQ.0)THEN
C                  * no. equation - var. is designer controlled *
                   GOTO 79
                ENDIF
             ENDIF
             IF(LCALC(LEVEL).GT.2)THEN
C               * are returning to this level after unstacking *
C               * check calculation stack pointers agree - this is a
C               * systems check to alert to any pgming errors *
                IF(LSTKCK(LEVEL).NE.LSTACK)THEN
C                  * systems error *
                   WRITE(*,92)LSTACK,LSTKCK(LEVEL),LEVEL
                   WRITE(10,92)LSTACK,LSTKCK(LEVEL),LEVEL
                   WRITE(11,92)LSTACK,LSTKCK(LEVEL),LEVEL
                   STOP
                ENDIF
             ENDIF
             WRITE(10,9906)
             DO 43 KEQN=LCALC(LEVEL),LEQN(LEVEL,1)+1
                WRITE(10,9907)LEQUAT(LEVEL),KEQN,LEQN(LEVEL,KEQN)
                IF(LEQN(LEVEL,KEQN).GT.50)THEN
C                  * eqn element is a var address *
                   IF(NPRD(1,LEQN(LEVEL,KEQN)).NE.0)THEN
C                     * data present, put on stack *
                      LSTACK=LSTACK+1
                      WRITE(10,9903)LEQN(LEVEL,KEQN),LSTACK
                      STACK(LSTACK)=RCASE(1,LEQN(LEVEL,KEQN))
                   ELSE
C                     * data not present - stack *
                      READ(12,106,REC=LEQN(LEVEL,KEQN))NEQN,NTABD
```

```
                  NTABLE(LEVEL+1)=NTABD
                  MISING(LEVEL+1)=LEQN(LEVEL,KEQN)
                  LEQUAT(LEVEL+1)=NEQN
                  LCALC(LEVEL)=KEQN
                  LSTKCK(LEVEL)=LSTACK
                  LDCVDT(LEVEL+1)=LDCVDT(LEVEL)
                  GOTO 71
               ENDIF
            ELSEIF(LEQN(LEVEL,KEQN).LE.35)THEN
C              * operator - carry out calculation *
               IF(LEQN(LEVEL,KEQN).EQ.1)THEN
C                 * logical operator '=' *
                  IF(STACK(LSTACK).EQ.STACK(LSTACK-1))THEN
                     STACK(LSTACK-1)=1
                  ELSE
                     STACK(LSTACK-1)=0
                  ENDIF
                  LSTACK=LSTACK-1
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.2)THEN
C                 * logical operator '<' *
                  IF(STACK(LSTACK-1).LT.STACK(LSTACK))THEN
                     STACK(LSTACK-1)=1
                  ELSE
                     STACK(LSTACK-1)=0
                  ENDIF
                  LSTACK=LSTACK-1
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.3)THEN
C                 * logical operator '>' *
                  IF(STACK(LSTACK-1).GT.STACK(LSTACK))THEN
                     STACK(LSTACK-1)=1
                  ELSE
                     STACK(LSTACK-1)=0
                  ENDIF
                  LSTACK=LSTACK-1
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.5)THEN
C                 * subtraction *
                  LSTACK=LSTACK-1
                  STACK(LSTACK)=STACK(LSTACK)-STACK(LSTACK+1)
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.6)THEN
C                 * addition *
                  LSTACK=LSTACK-1
                  STACK(LSTACK)=STACK(LSTACK)+STACK(LSTACK+1)
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.7)THEN
C                 * multiplication *
                  LSTACK=LSTACK-1
                  STACK(LSTACK)=STACK(LSTACK)*STACK(LSTACK+1)
C
               ELSEIF(LEQN(LEVEL,KEQN).EQ.8)THEN
C                 * division *
```

```
            LSTACK=LSTACK-1
            IF(STACK(LSTACK+1).NE.0)THEN
               STACK(LSTACK)=STACK(LSTACK)/STACK(LSTACK+1)
            ELSE
               WRITE(*,93)
               WRITE(10,93)
               WRITE(11,93)
               NERROR=1
               RETURN
            ENDIF
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.9)THEN
C              * exponentiation *
               LSTACK=LSTACK-1
               STACK(LSTACK)=STACK(LSTACK)**STACK(LSTACK+1)
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.12)THEN
C              * sin(x) - note x is in radians *
               STACK(LSTACK)=SIN(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.13)THEN
C              * cos(x) - note x is in radians *
               STACK(LSTACK)=COS(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.14)THEN
C              * tan(x) - note x is in radians *
               STACK(LSTACK)=TAN(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.15)THEN
C              * asin(x) - note resultant is in radians *
               STACK(LSTACK)=ASIN(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.16)THEN
C              * acos(x) - note resultant is in radians *
               STACK(LSTACK)=ACOS(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.17)THEN
C              * atan(x) - note resultant is in radians *
               STACK(LSTACK)=ATAN(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.18)THEN
C              * natural log *
               STACK(LSTACK)=LOG(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.19)THEN
C              * log 10 *
               STACK(LSTACK)=LOG10(STACK(LSTACK))
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.20)THEN
C              * pi *
               LSTACK=LSTACK+1
               STACK(LSTACK)=4*ATAN(1.0)
C
         ELSEIF(LEQN(LEVEL,KEQN).EQ.22)THEN
```

```
C                    * logical TRUE or YES*
                     LSTACK=LSTACK+1
                     STACK(LSTACK)=1
C
                  ELSEIF(LEQN(LEVEL,KEQN).EQ.23)THEN
C                    * logical FALSE or NO *
                     LSTACK=LSTACK+1
                     STACK(LSTACK)=0
C
                  ELSE
C                    * system error - illegal element in equation *
                     WRITE(*,94)NADRES,LEQUAT(LEVEL),LEQN(LEVEL,KEQN),
     +                  (LEQN(LEVEL,J),J=1,LEQN(LEVEL,1)+1)
                     WRITE(11,94)NADRES,LEQUAT(LEVEL),LEQN(LEVEL,KEQN),
     +                  (LEQN(LEVEL,J),J=1,LEQN(LEVEL,1)+1)
                     STOP
                  ENDIF
               ELSE
C                 * eqn element is a look-up table number *
                  LKNO=LEQN(LEVEL,KEQN)-35
                  DO 45 KDIM=1,LKDIM(LKNO)
                     NCOORD(KDIM)=STACK(LSTACK-LKDIM(LKNO)+KDIM)
   45             CONTINUE
                  CALL LKPOSN(LKNO,LKDIM,LKSIZE,LKPTR,NCOORD,LOCATE)
                  LSTACK=LSTACK-LKDIM(LKNO)+1
                  STACK(LSTACK)=RLKDAT(LOCATE)
                  WRITE(10,9905)RLKDAT(LOCATE),LSTACK
               ENDIF
               WRITE(10,9995)LEQUAT(LEVEL),KEQN,LSTACK
   43       CONTINUE
C           * put calculated data into work area
C           * and set presence of data flag *
            RCASE(1,NADRES)=STACK(LSTACK)
            NPRD(1,NADRES)=1
            LSTACK=LSTACK-1
            LSTKCK(LEVEL)=LSTACK
            WRITE(10,9904)RCASE(1,NADRES),NADRES,LSTACK
            LASTEQ(1,NADRES)=LEQUAT(LEVEL)
         ENDIF
      ENDIF
C
C     * unstacking *
   72 IF(LEVEL.EQ.1)THEN
         WRITE(10,9902)
         RETURN
      ELSE
         LEVEL=LEVEL-1
         WRITE(10,9993)LEVEL,lstack
         NSFLAG=0
C        * return to previously suspended level *
         GOTO 77
      ENDIF
C     * end of unstacking segment *
C
```

```
 91 FORMAT(/1x/' ERROR during execution of table ',A,' no.',I2
   +          /'       No rule applies.  Run aborted.')
 92 FORMAT(' [subroutine TABEX3 systems error.'
   +       /' Calculation stack parameters disagree.'
   +       /' LSTACK = ',I2,' LSTKCK(LEVEL) = ',I2,' LEVEL = ',I2,
   +        ' RUN ABORTED]')
 93 FORMAT(/1X/' ERROR attempt to divide by zero')
 94 FORMAT(' [subroutine TABEX3 systems error.'
   +       /'  illegal element in reverse-polish equation.'
   +       /'  Partial listing follows ... '
   +       /' ',15I4' ]')
 95 FORMAT(' [subroutine TABEX3 error.'
   +       /'  non-logical value of a condition stub found.'
   +       /'  table no. ',I3,' condition no. ',I2,' stub value ',I3
   +      //'  RUN ABORTED ]')
106 FORMAT(I4,1X,I4)
107 FORMAT(//' The current value of ',A,' is ',3P1E12.2E2,' ',A
   +       /' If you wish to change this value enter the new value '
   +       /' (up to 12 digits including a compulsory decimal point'
   +       /' and optional 2 digit exponent) and press <RETURN>'
   +       /' If no change required press <RETURN> with no entry'
   +    //' :: enter new value or press <RETURN> '$)
1071 FORMAT(//' The current value of ',A,' is ',A
   +       /' If you wish to change this value enter the new value '
   +       /' (either Y or N) and press <RETURN>'
   +       /' If no change required press <RETURN> with no entry'
   +    //' :: enter new value or press <RETURN> '$)
108 FORMAT(/1x/' [table ',A,', table no. ',I2,', rule no. ',I2
   +           ' applies]'
   +        /1x,A,' is currently undefined.')
109 FORMAT(/1x/' [table ',A,', table no. ',I2,', rule no. ',I2
   +           ' applies]'
   +        /1x,A,'= ',3P1E12.2E2,' ',A)
110 FORMAT(/1x/' [table ',A,', table no. ',I2,', rule no. ',I2
   +           ' applies]'
   +        /1x,A)
111 FORMAT(/1x/' [table ',A,', table no. ',I2,', rule no. ',I2
   +           ' applies]'/1X)
201 FORMAT(10X,A,1X,A)
202 FORMAT(E12.2E2)
203 FORMAT(82X,80I4)
204 FORMAT(I4,1X,I4,1X,A,1X,A,I4)
205 FORMAT(A)
9901 FORMAT(/' TABEX3 called, d.t. ',I2,' to be executed.'//)
9902 FORMAT(' Returning from TABEX3'///)
9903 FORMAT(' Data at address ',I4,' present,'
   +         /' to calc stack at height ',I2)
9904 FORMAT(/' Data value ',3P1E12.2E2
   +        /' case address ',I4,' transferred from stack.'
   +        /' Calculation stack height reduced to ',I2/)
9905 FORMAT(' Look-up data ',3P1E12.2E2,' onto stack at height ',I2)
9906 FORMAT(//' About to enter loop 43'/)
9907          FORMAT(/' Top of loop 43. Eqn no.= ',I2
   +             /' KEQN is at position ',I2,' Element is ',I2)
```

```
9991 FORMAT(//' stacking to level = ',i2)
9992 FORMAT(1X,' ntnum=',i2)
9993 FORMAT(//' unstacking to level = ',i2,' lstack=',i2)
9994 FORMAT(1X,' nadres=',i4,' lstack=',i2,' lstkck=',i2)
9995 FORMAT(1X,' End of loop 43. Eqn no.= ',i2,' keqn=',i2,
    +' lstack=',i2)
     END
```

```
      SUBROUTINE UNDEPN(NXINGD,LOCATE)
C*****************************************************************************
C   PURPOSE :    To remove an address from a dependence list after eqn
C                modification has changed dependence relationships.
C   METHOD :     NXINGD's dep. list is scanned for LOCATE which is
C                removed when found.
C   ARGUMENTS :
C                NXINGD  address of a previous ingredient of LOCATE.
C                LOCATE  address of var. whose eqn has been modified.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C*****************************************************************************
      CHARACTER HOLD*41
      INTEGER NDEPN(50),NSORCE(50)
C
      READ(12,101,REC=NXINGD)HOLD,(NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
C     * find LOCATE on NXINGD's dependents list *
      DO 41 KPOINT=2,NDEPN(1)+1
         IF(NDEPN(KPOINT).EQ.LOCATE)THEN
C           * remove LOCATE from list *
            NDEPN(1)=NDEPN(1)-1
            NSORCE(1)=NSORCE(1)-1
            DO 42 KREST=KPOINT,NDEPN(1)
               NDEPN(KREST)=NDEPN(KREST+1)
               NSORCE(KREST)=NSORCE(KRESET+1)
   42       CONTINUE
         ENDIF
   41 CONTINUE
C     * write updated list of dependents back to file *
      WRITE(12,101,REC=NXINGD)HOLD,(NDEPN(J),J=1,50),(NSORCE(K),K=1,50)
      RETURN
C
  101 FORMAT(A41,50I4,50I4)
      END
```

```
      SUBROUTINE WARN3(NFIRST,NWFLAG)
C*****************************************************************
C   PURPOSE :  To set the presence of data flags of all data affected
C              by a change in a variable value to zero.
C   METHODS :  Dependence lists are used to trace all dependent data.
C              Trans table dependences are also traced.
C   ARGUMENTS :
C              NFIRST  address of variable originally altered.
C              NWFLAG  indicates whether all case studies are to be
C                      warned (=1) or just the work area (=0).
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL ARRAYS :
C              NADRES  current address at each level.
C              LTABLE  decision table being scanned at each level.
C              LACTNO  records current action in decision table LTABLE.
C              LDFLAG  indicates whether a list of decision tables is
C                      being scanned. (ie dep. list of a condition).
C   LOCAL VARIABLES :
C              NSFLAG  =1 if current level of operations was reached
C                      by stacking.
C              NFLAG   =0 if no dependents of NFIRST are reset.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  1st entry in dependence list gives lsit length.
C              Priority is given to processing dependents existing at
C              deeper levels in the data network. When the dependent
C              of a variable is seen to also have dependents,
C              processing of the current list is suspended, stacking
C              occurs and warning of data at the deeper level begins.
C              Unstacking later returns processing to the previously
C              suspended level. Dependents of a variable are only
C              warned if the variable's NPRD=1.
C*****************************************************************
      INTEGER NDEPN(20,50),LDEPN(20),NADRES(20),LTABLE(20),LACTNO(20),
     +        LDFLAG(20),NSORCE(20,50)
      CHARACTER NAME*20, TABLE
      CHARACTER*30 MESAGE(100)
C
      COMMON/BLOK1/NCOND(100),NACT(100),NRULE(100),
     +   NTYPE(100,50),NSTORE(100,50),METHOD(100,50)
      COMMON/BLOK1A/TABLE(100,50,50),MESAGE
      COMMON/BLOK7/NPRD(6,100),RCASE(6,100),LASTRL(6,100),
     +   NRESET(50),LASTEQ(6,500)
C
      NFLAG=0
      LEVEL=0
      NADRES(1)=NFIRST
      LTABLE(1)=0
      LDFLAG(1)=0
C
C     * stacking segment *
   51 LEVEL=LEVEL+1
      LDEPN(LEVEL)=2
      LACTNO(LEVEL)=1
```

```
      NSFLAG=1
C     * end of stacking segment *
C
   52 IF(LTABLE(LEVEL).GT.0)THEN
C        * trans-table dependence relationships exist *
C        * scanning of table action entries and stubs *
         LA=LACTNO(LEVEL)
         NTAB=LTABLE(LEVEL)
         LAEND=NACT(NTAB)
C        * check table has been run *
         IF(LASTRL(1,NTAB).NE.0)THEN
C           * check that cond entry corresponding to cond was non '-' *
            DO 44 KCOND=1,NCOND(NTAB)
               IF(NSTORE(NTAB,KCOND).EQ.NADRES(LEVEL))THEN
C                 * have found affected condition *
C                 * check for immaterial condition entry at last rule *
                  IF(TABLE(NTAB,KCOND,LASTRL(1,NTAB)).EQ.'-')THEN
C                    * affected condition had no bearing on last rule *
                     GOTO 54
                  ELSE
                     GOTO 71
                  ENDIF
               ENDIF
   44       CONTINUE
C           * check for any dependent rules *
   71       DO 43 KACT=LA,LAEND
               KROW=NCOND(NTAB)+KACT
               IF(TABLE(NTAB,KROW,LASTRL(1,NTAB)).EQ.'X'.AND.
     +            NTYPE(NTAB,KACT).EQ.1)THEN
C                 * action applies and is variable calculation *
C                 * cond stub change may mean rule no longer applies *
                  NADRES(LEVEL+1)=NSTORE(NTAB,KROW)
                  LTABLE(LEVEL+1)=0
                  LACTNO(LEVEL)=KACT+1
                  LDFLAG(LEVEL+1)=0
                  NPRD(1,NSTORE(NTAB,KROW))=0
                  IF(NWFLAG.EQ.1)THEN
C                    * all case studies to be WARNed *
                     DO 421 KCASE=2,6
                        NPRD(KCASE,NEXT)=0
  421                CONTINUE
                  ENDIF
                  GOTO 51
               ENDIF
   43       CONTINUE
         ENDIF
C
      ELSE
C        * scanning a dependence list *
         IF(LDFLAG(LEVEL).EQ.1)THEN
C           * are scanning a dependents list associated with a condition
C           * stub - list will consist of decision table numbers *
C           * cond stub dep list starts 1 further into DEPN() than
C           * an ordinary variable dependence list *
```

```
            LDPEND=NDEPN(LEVEL,2)+2
            IF(NSFLAG.EQ.1)THEN
               LD=LDEPN(LEVEL)+1
            ELSE
               LD=LDEPN(LEVEL)
            ENDIF
            IF(LD.LE.LDPEND)THEN
C              * haven't completed list yet - prepare to stack *
               LDEPN(LEVEL)=LD+1
               LACTNO(LEVEL)=0
               NADRES(LEVEL+1)=NADRES(LEVEL)
               LTABLE(LEVEL+1)=NDEPN(LEVEL,LD)
               GOTO 51
            ENDIF
         ELSE
C           * ordinary dependence list *
            NPLACE=NADRES(LEVEL)
            IF(NSFLAG.EQ.1)THEN
C              * read in dependence & source lists for this level *
               READ(12,101,REC=NPLACE)(NDEPN(LEVEL,J),J=1,50),
     +            (NSORCE(LEVEL,K),K=1,50)
            ENDIF
            NDPEND=NDEPN(LEVEL,1)+1
            LD=LDEPN(LEVEL)
            DO 41 KDEPN=LD,NDPEND
C              * identify next dependent at address NPLACE *
               NEXT=NDEPN(LEVEL,KDEPN)
               IF(NPRD(1,NEXT).NE.1.OR.
     +            NSORCE(LEVEL,KDEPN).NE.LASTEQ(1,NEXT))GOTO 41
C              * set presence of data flag at NEXT to void *
               NPRD(1,NEXT)=0
               IF(NWFLAG.EQ.1)THEN
C                 * all case studies to be WARNed *
                  DO 42 KCASE=2,6
                     NPRD(KCASE,NEXT)=0
   42             CONTINUE
               ENDIF
C              * display list of variables affected by the change *
               READ(12,19,REC=NEXT)NAME
               IF(NFLAG.EQ.0)THEN
C                 * NAME is first data affected *
                  WRITE(*,18)
                  NFLAG=1
               ENDIF
               WRITE(*,17)NAME
C              * are dependents indicated at address next? *
   53          READ(12,101,REC=NPLACE)NDEPEN
               IF(NDEPEN.NE.0)THEN
                  LDEPN(LEVEL)=KDEPN+1
                  NADRES(LEVEL+1)=NEXT
                  LDFLAG(LEVEL+1)=0
                  LTABLE(LEVEL+1)=0
                  GOTO 51
               ELSE
```

```
C                  * could be a cond. stub dependence list *
                   IF(NDEPN(LEVEL,2).NE.0)THEN
C                     * condition stub at NEXT *
                      LDEPN(LEVEL)=KDEPN+1
                      NADRES(LEVEL+1)=NEXT
                      LDFLAG(LEVEL+1)=0
                      LTABLE(LEVEL+1)=0
                      GOTO 51
                   ENDIF
                ENDIF
   41        CONTINUE
          ENDIF
       ENDIF
C
C      * unstacking segment *
   54  IF(LEVEL.EQ.1)THEN
          IF(NFLAG.EQ.0)WRITE(*,20)
          RETURN
       ENDIF
       LEVEL=LEVEL-1
       NSFLAG=0
       GOTO 52
C      * end of unstacking segment *
C
   17  FORMAT(1X,A)
   18  FORMAT(' As a result of this change the values of the'
     +       /1X,' following variables have been set to void - ')
   19  FORMAT(21X,A)
   20  FORMAT(/1X/' Any data which would''ve been affected by this '
     +           'change'/' has already been set to void '/)
  101  FORMAT(41X,50I4,50I4)
       END
```

```
      SUBROUTINE XINDEX(LAST,STRING,NPT1,NPT2,NADRES,NFLAG)
C*****************************************************************
C   PURPOSE :  To identify the storage locations of existing variables
C              in DIRECT.dat (the directory) and allocate storage
C              locations to new variables.
C   METHODS :  Linear search.
C   ROUTINES CALLED :
C              INSERT
C   ARGUMENTS :
C              LAST    indicates whether (>0) or not (=0) an unknown
C                      name is to be allocated a storage location.
C                      (Allows use of XINDEX in a querying or
C                      an indexing role).
C              STRING  input equation from ENCODE.
C              NPT1    position of first character of name in STRING.
C              NPT2    position of last character of name in STRING.
C              NADRES  existing or allocated address of a variable.
C              NFLAG   =1 if name already known.
C   COMMON ARRAYS AND VARIABLES : see DESIGN for a full list.
C   LOCAL VARIABLES :
C              NBEGIN  beginning location of block of names of same
C                      length as queried name.
C              NEND    end location of block of names of same
C                      length as queried name.
C   PROGRAMMER : A.A.Hunt
C   VERSION / DATE : 1.00 /  1- 6-88
C   REMARKS :  DIRECT.dat stores names in order of name length then
C              alphabetical order.
C*****************************************************************
      CHARACTER*20 NAME,WORD,BLANK
      CHARACTER STRING(80)
      COMMON/BLOK4/NENTRY,NPLACE,NDETAB,NXTAB,NMESGE,NUMEQN,NLKUP,LKTODO
C     * remove leading blanks from name *
      NSTART=NPT1
      NEND=NPT2
      DO 42 K42=NSTART,NPT2
         IF(STRING(K42).NE.' ')THEN
            NPT1=K42
            GOTO 72
         ENDIF
   42 CONTINUE
C     * remove trailing blanks from name *
   72 DO 43 K43=NPT1,NEND
         IF(STRING(K43).NE.' ')THEN
            NPT2=K43
         ENDIF
   43 CONTINUE
      LENGTH=NPT2-NPT1+1
C     * following read/write necessary through lack of
C     * character substring handling facility *
      WRITE(12,10,REC=2)(STRING(J),J=NPT1,NPT2)
      READ(12,14,REC=2)NAME
C
      READ(13,11,REC=LENGTH)NBEGIN,NEND
```

```
      NSIZE=NEND-NBEGIN
C     * linear search *
      DO 41 K41=NBEGIN,NEND
         READ(13,12,REC=K41)NADRES,WORD
         IF(NAME.EQ.WORD)THEN
            NFLAG=1
            RETURN
         END IF
         IF(LLT(NAME,WORD))THEN
            IF(LAST.EQ.0)THEN
C              * NAME isn't to be asigned a place *
               WRITE(*,13)NAME
C              * NADRES=0 indicates to DESRUN that NAME not known *
               NADRES=0
               NFLAG=0
               RETURN
            ELSE
C              * insert NAME into list *
               CALL INSERT(LENGTH,NAME,K41,NADRES)
               NFLAG=0
               RETURN
            END IF
         END IF
   41 CONTINUE
C     * NAME to be added after last in list *
      NARG=NEND+1
      CALL INSERT(LENGTH,NAME,NARG,NADRES)
      NFLAG=0
      RETURN
C
   10 FORMAT(20A1)
   11 FORMAT(I4,1X,I4)
   12 FORMAT(I4,1X,A20)
   13 FORMAT(' ERROR : ',A,' can not be found in the list of variables')
   14 FORMAT(A)
      END
```

APPENDIX C

WORKED EXAMPLE

This appendix demonstrates the application of the Program Generator to a beam bending problem and consists of the screen output and user responses from the demonstration terminal session. User responses are hand written so that they may be readily distinguished. Formulae are from Roarke (1975).

To demonstrate the use of look-up tables the table RHS is used to hold selected steel cross-sectional data for two steel sections.

The demonstration shows the designer entering and then running the beam design program. The first run is unsuccessful and leads to the designer changing steel sections and opting for a material having a higher yield stress than that used in the first run. Page C-32 shows the contents of the output file at the conclussion of the successful program run.

The top right-hand corner of each page is annotated to indicate the program generation and each of the two run phases of the demonstration.

A PROGRAM GENERATOR FOR DESIGNERS

Version 1.0 1988

Mechanical Engineering Department

School Of Engineering
University Of Canterbury

Ensure shift lock is on (capital letters are required)!!
:: are you ready to continue? (Y) Y↵

MAIN MENU

Do you wish to :-

1 Run an existing program or view data, tables etc
2 Modify an existing program or data
3 Create a new program
4 Quit

:: enter number of choice and press <RETURN> 3↵

:: enter program title (up to 30 characters) -

BEAM BENDING↵
Decision table BEAM BENDING
is now to be entered. Conditions and actions
may not exceed 50 in total.

:: enter number of conditions - 1↵
:: enter number of actions - 5↵
:: enter number of rules - 2↵
Entry of decision table conditions.

Conditions must not be more than 20 characters long and
must not contain operators other than < > or =

```
:: enter condition number  1, no more than 20 characters long -
SB < SY ↵

Decision table actions are now to be entered.


Is action number  1 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 4 ↵

:: enter number of message, 0 if one not yet listed - 0
:: enter message, up to 30 characters long -
BENDING STRESS ↵

Is action number  2 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 2 ↵

:: enter variable name, up to 20 characters long -
SB ↵

Is action number  3 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1 ↵

:: enter variable name, up to 20 characters long -
MS ↵
:: enter units for MS
   up to 10 characters or press <RETURN> if dimensionless -
↵

:: enter RHS of equation defining MS
   no more than 70 characters -
SY/SB-ONE

Is action number  4 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message
```

```
:: enter action type number - 2↵

:: enter variable name, up to 20 characters long -
MS↵

Is action number  5 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 4↵
          1       BENDING STRESS OK

:: enter number of message, 0 if one not yet listed - 0↵
:: enter message, up to 30 characters long -
BEAM OVERSTRESSED↵
Entry of rule  1
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.


     SB<SY                ? || Y↵
==============================
say  BENDING STRESS OK      || X↵
show SB                     || X↵
calc MS                     || X↵
show MS                     || X↵
say  BEAM OVERSTRESSED      ||↵

Entry of rule  2
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.


     SB<SY                ? || YN
==============================
say  BENDING STRESS OK      || X↵
show SB                     || XX↵
calc MS                     || XX↵
show MS                     || XX↵
say  BEAM OVERSTRESSED      ||  X↵


:: enter units for SB
   up to 10 characters or press <RETURN> if dimensionless -
MPA↵
```

```
Is SB                    to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 2 ↵


:: enter RHS of equation defining SB
   no more than 70 characters -
M*Y/I ↵

:: enter units for SY
   up to 10 characters or press <RETURN> if dimensionless -
MPA ↵
Is SY                    to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for ONE
   up to 10 characters or press <RETURN> if dimensionless -
↵
Is ONE                   to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for M
   up to 10 characters or press <RETURN> if dimensionless -

Is M                     to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 1 ↵
:: enter table title (up to 30 characters) -

BENDING MOMENT ↵
```

```
INFORMATION :- variables whose values are entered directly by the user
are automatically associated with any decision table which
directly or indirectly uses them and, unless it is specified
otherwise, a value will be requested for the variable each
time the table is used. If the value of a relevant variable is
not being queried, either: use the scratch case study (case 1)
and execute the table, or use the query-don't query option
on the RUN MENU and then execute the table.




:: enter units for Y
   up to 10 characters or press <RETURN> if dimensionless -
M↵
Is Y                        to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 2↵


:: enter RHS of equation defining Y
   no more than 70 characters -
RHS(SECTION NUMBER, TWO)↵

:: enter units for I
   up to 10 characters or press <RETURN> if dimensionless -
M↵
Is I                        to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 2↵


:: enter RHS of equation defining I
   no more than 70 characters -
RHS(SECTION NUMBER, ONE)↵
:: enter units for SECTION NUMBER
   up to 10 characters or press <RETURN> if dimensionless -
↵
```

```
Is SECTION NUMBER       to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3↵


:: enter units for TWO
   up to 10 characters or press <RETURN> if dimensionless -
↵
Is TWO                  to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3↵


Decision table  BENDING MOMENT
is now to be entered. Conditions and actions
may not exceed 50 in total.

:: enter number of conditions - 6 ↵
:: enter number of actions    - 5 ↵
:: enter number of rules      - 6 ↵
Entry of decision table conditions.

Conditions must not be more than 20 characters long and
must not contain operators other than < > or =


:: enter condition number  1, no more than 20 characters long -
LEFT END FREE ↵
:: enter condition number  2, no more than 20 characters long -
LEFT END FIXED ↵
:: enter condition number  3, no more than 20 characters long -
LEFT END GUIDED ↵
:: enter condition number  4, no more than 20 characters long -
LEFT END SS ↵
:: enter condition number  5, no more than 20 characters long -
RIGHT END FIXED ↵
:: enter condition number  6, no more than 20 characters long -
RIGHT END SS ↵


Decision table actions are now to be entered.
```

```
Is action number  1 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1↵

:: enter variable name, up to 20 characters long -
M↵

:: enter RHS of equation defining M
   no more than 70 characters -
W*L ↵

Is action number  2 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1↵

:: enter variable name, up to 20 characters long - M↵
The following are the existing equations for the
calculation of M

      6 W*L


:: enter desired equation number, 0 if a newequation is required - 0↵


:: enter RHS of equation defining M
   no more than 70 characters -
W*L/2 ↵

Is action number  3 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1 ↵

:: enter variable name, up to 20 characters long -
M ↵
The following are the existing equations for the
calculation of M

      6 W*L
      7 W*L/TWO


:: enter desired equation number, 0 if a newequation is required - 0 ↵
```

```
:: enter RHS of equation defining M
   no more than 70 characters -
CONST * W * L ↵
Is action number  4 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1 ↵

:: enter variable name, up to 20 characters long -
M ↵
The following are the existing equations for the
calculation of M

      6 W*L
      7 W*L/TWO
      8 W*L*CONST


:: enter desired equation number, 0 if a newequation is required - 0 ↵


:: enter RHS of equation defining M
   no more than 70 characters -
W * L / EIGHT ↵
Is action number  5 to consist of :-
      1  calculation of a variable
      2  displaying the value of a variable
      3  executing a decision table
      4  displaying a message

:: enter action type number - 1 ↵

:: enter variable name, up to 20 characters long -
M ↵
The following are the existing equations for the
calculation of M

      6 W*L
      7 W*L/TWO
      8 W*L*CONST
      9 W*L/EIGHT


:: enter desired equation number, 0 if a newequation is required - 0 ↵


:: enter RHS of equation defining M
   no more than 70 characters -
W * L / FOUR ↵
```

```
Entry of rule  1
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.


     LEFT END FREE        ? ||Y↵
     LEFT END FIXED       ? ||N↵
     LEFT END GUIDED      ? ||N↵
     LEFT END SS          ? ||N↵
     RIGHT END FIXED      ? ||Y↵
     RIGHT END SS         ? ||N↵
==================================
calc M                      ||X↵
calc M                      ||↵
calc M                      ||↵
calc M                      ||↵
calc M                      ||↵

Entry of rule  2
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.


     LEFT END FREE        ? || Y N ↵
     LEFT END FIXED       ? || N N ↵
     LEFT END GUIDED      ? || N Y ↵
     LEFT END SS          ? || N N ↵
     RIGHT END FIXED      ? || Y Y ↵
     RIGHT END SS         ? || N N ↵
==================================
calc M                      || X ↵
calc M                      ||   X↵
calc M                      ||  ↵
calc M                      ||  ↵
calc M                      ||  ↵

Entry of rule  3
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.
```

```
     LEFT END FREE        ? || YN N↵
     LEFT END FIXED       ? || NN N↵
     LEFT END GUIDED      ? || NY N↵
     LEFT END SS          ? || NN Y↵
     RIGHT END FIXED      ? || YY Y↵
     RIGHT END SS         ? || NN N↵
==================================
calc M                      || X ↵
calc M                      ||  X↵
calc M                      ||   X↵
calc M                      ||   ↵
calc M                      ||   ↵
```

Entry of rule 4
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.

```
     LEFT END FREE        ? || YNN N↵
     LEFT END FIXED       ? || NNN Y↵
     LEFT END GUIDED      ? || NYN N↵
     LEFT END SS          ? || NNY N↵
     RIGHT END FIXED      ? || YYY Y↵
     RIGHT END SS         ? || NNN N↵
==================================
calc M                      || X  ↵
calc M                      ||  X ↵
calc M                      ||   X↵
calc M                      ||    X↵
calc M                      ||    ↵
```

Entry of rule 5
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.

```
     LEFT END FREE        ? || YNNN N↵
     LEFT END FIXED       ? || NNNY N↵
     LEFT END GUIDED      ? || NYNN N↵
     LEFT END SS          ? || NNYN Y↵
     RIGHT END FIXED      ? || YYYY N↵
     RIGHT END SS         ? || NNNN Y↵
==================================
calc M                      || X   ↵
calc M                      ||  X  ↵
calc M                      ||   X ↵
calc M                      ||    X↵
calc M                      ||     X↵
```

```
Entry of rule  6
  Condition entries may be Y,N or -
  Action entries may be X or blank.

Make entry when cursor stops at each row.


     LEFT END FREE        ? || YNNNN N ↵
     LEFT END FIXED       ? || NNNYN N ↵
     LEFT END GUIDED      ? || NYNNN Y ↵
     LEFT END SS          ? || NNYNY N ↵
     RIGHT END FIXED      ? || YYYYN N ↵
     RIGHT END SS         ? || NNNNY Y ↵
=====================================
calc M                      || X    X ↵
calc M                      ||  X   ↵
calc M                      ||   X  ↵
calc M                      ||    X ↵
calc M                      ||     X↵

Is LEFT END FREE         to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3↵

Is LEFT END FIXED        to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3 ↵

Is LEFT END GUIDED       to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3↵

Is LEFT END SS           to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3 ↵
```

```
Is RIGHT END FIXED      to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3 ↵

Is RIGHT END SS         to be assigned a value by -

1  the outcome of a decision table
2  (option 2 not valid for a condition expression)
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for W
   up to 10 characters or press <RETURN> if dimensionless -
N↵
Is W                    to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 2↵


:: enter RHS of equation defining W
   no more than 70 characters -
MASS * G ↵

:: enter units for L
   up to 10 characters or press <RETURN> if dimensionless -
M↵
Is L                    to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for CONST
   up to 10 characters or press <RETURN> if dimensionless -
↵
```

```
Is CONST                to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for EIGHT
   up to 10 characters or press <RETURN> if dimensionless -
↵
Is EIGHT                to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for FOUR
   up to 10 characters or press <RETURN> if dimensionless -
↵
Is FOUR                 to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for MASS
   up to 10 characters or press <RETURN> if dimensionless -
  KG ↵
Is MASS                 to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵


:: enter units for G
   up to 10 characters or press <RETURN> if dimensionless -
MS^-2
```

```
Is G                        to be assigned a value by -

1  the outcome of a decision table
2  the result of an equation
3  as a direct input from the designer

:: enter number of choice - 3 ↵

Look-up table RHS                    is now to be processed

:: enter title of axis number 1 - SECTION NUMBER ↵

:: enter number of entries along axis number 1 - 2 ↵

:: enter title of axis number 2 - PROPERTY NUMBER ↵

:: enter number of entries along axis number 2 - 2 ↵

Data is now to be loaded into look-up table RHS
:: enter table entry in turn below the DATA heading,
   up to 12 digits including a decimal point and
   optional 2 digit exponent.

SECTION NUMBER      PROPERTY NUMBER     DATA
 1                   1                  1310.0E-08 ↵
SECTION NUMBER      PROPERTY NUMBER     DATA
 1                   2                  76·0E-03 ↵
SECTION NUMBER      PROPERTY NUMBER     DATA
 2                   1                  1830·0E-08 ↵
SECTION NUMBER      PROPERTY NUMBER     DATA
 2                   2                  76·0E-03 ↵

Data for this look-up table is now fully loaded
```

MAIN MENU

Do you wish to :-

1 Run an existing program or view data, tables etc
2 Modify an existing program or data
3 Create a new program
4 Quit

:: enter number of choice and press <RETURN> 4 ↵

To obtain your printout of output generated this session type PRINT OUTPUT.DAT when prompt appears.

The output file will be DELETED on commencing the next run

PROGRAM RUN 1

A PROGRAM GENERATOR FOR DESIGNERS

Version 1.0 1988

Mechanical Engineering Department

School Of Engineering
University Of Canterbury

Ensure shift lock is on (capital letters are required)!!
:: are you ready to continue? (Y) Y↵

MAIN MENU

Do you wish to :-

1 Run an existing program or view data, tables etc
2 Modify an existing program or data
3 Create a new program
4 Quit

:: enter number of choice and press <RETURN> 1 ↵

RUN MENU

Do you wish to -

1 select and run a program
2 re-run the last program
3 select a case study
4 save current case study
5 change variable status to query/don't query each run
6 preset (override) the value of a variable normally
calculated by the program. (Resets after next run)
Select your case study BEFORE using override.
7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry
0 return to main menu

:: enter number of choice - 3 ↵

```
A list of case studies will be displayed. A copy of
the case study you selected will be copied into the work area
The original data remains intact and will only be modified if
you choose to overwrite it later. (menu option 4)


case 1      work area                    .
case 2      Vacant                       .
case 3      Vacant                       .
case 4      Vacant                       .
case 5      Vacant                       .
case 6      Vacant                       .


:: enter case number of your choice,
   (select 1 if you wish to start from scratch) - 1 ↵


RUN MENU
--------

Do you wish to -

1  select and run a program
2  re-run the last program
3  select a case study
4  save current case study
5  change variable status to query/don't query each run
6  preset (override) the value of a variable normally
   calculated by the program. (Resets after next run)
   Select your case study BEFORE using override.
7  view a decision table
        or the current value of a variable
        or all equations associated with a variable
        or a look-up table entry
0  return to main menu

:: enter number of choice - 1 ↵


The following is a list of all decision tables :-

table  1     BEAM BENDING
table  2     BENDING MOMENT


:: enter number of decision table - 1 ↵
```

```
The current value of LEFT END FREE        is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END FIXED       is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END GUIDED      is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END SS          is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> Y↵


Any data which would've been affected by this change
has already been set to void


The current value of RIGHT END FIXED      is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵
```

PROGRAM RUN 1

```
The current value of RIGHT END SS          is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> Y↵


Any data which would've been affected by this change
has already been set to void


[table BENDING MOMENT                    , table no.  2, rule no.  5 applies]



The current value of MASS                  is     000.E+00 KG
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 5000.0 ↵


Any data which would've been affected by this change
has already been set to void



The current value of G                     is     000.E+00 MS^-2
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 9.81 ↵


Any data which would've been affected by this change
has already been set to void
```

PROGRAM RUN 1

```
The current value of L                    is     000.E+00 M
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 6.0 ↵


Any data which would've been affected by this change
has already been set to void



The current value of FOUR                 is     000.E+00
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 4.0 ↵


Any data which would've been affected by this change
has already been set to void



The current value of SECTION NUMBER       is     000.E+00
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 1.0 ↵


Any data which would've been affected by this change
has already been set to void



The current value of TWO                  is     000.E+00
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 2. ↵
As a result of this change the values of the
 following variables have been set to void -
M
```

PROGRAM RUN 1

```
The current value of ONE                    is     000.E+00
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 1.0↵


Any data which would've been affected by this change
has already been set to void



The current value of SY                     is     000.E+00 MPA
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 240.0E+06


Any data which would've been affected by this change
has already been set to void



[table BEAM BENDING                    , table no.  1, rule no.  2 applies]

SB                  =     430.E+06 MPA


[table BEAM BENDING                    , table no.  1, rule no.  2 applies]



[table BEAM BENDING                    , table no.  1, rule no.  2 applies]

MS                  =    -442.E-03


[table BEAM BENDING                    , table no.  1, rule no.  2 applies]

BEAM OVERSTRESSED



:: do you want the values of designer controlled  variables
written to the output file?  (Y/N) - N↵

Do you wish to DELETE the output from this LATEST
AND any PREVIOUS runs  (Y/N) - Y↵
```

RUN MENU

Do you wish to -

1 select and run a program
2 re-run the last program
3 select a case study
4 save current case study
5 change variable status to query/don't query each run
6 preset (override) the value of a variable normally
calculated by the program. (Resets after next run)
Select your case study BEFORE using override.
7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry
0 return to main menu

:: enter number of choice - 5 ↵

:: enter name of variable - G ↵

INFORMATION :- variables whose values are entered directly by the user are automatically associated with any decision table which directly or indirectly uses them and, unless it is specified otherwise, a value will be requested for the variable each time the table is used. If the value of a relevant variable is not being queried, either: use the scratch case study (case 1) and execute the table, or use the query-don't query option on the RUN MENU and then execute the table.

Is the value of the variable to be queried each run ? (Y/N) - N ↵

:: enter a set value for this variable
(up to 12 digits, including a compulsory decimal point and optional 2 digit exponent)
9.81 ↵
As a result of this change the values of the following variables have been set to void -
W
M
SB
SB<SY
MS

Note that you may vary the status of this variable from one case study to another

RUN MENU

Do you wish to -

1 select and run a program
2 re-run the last program
3 select a case study
4 save current case study
5 change variable status to query/don't query each run
6 preset (override) the value of a variable normally
calculated by the program. (Resets after next run)
Select your case study BEFORE using override.
7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry
0 return to main menu

:: enter number of choice - 5 ↵

:: enter name of variable - TWO ↵

INFORMATION :- variables whose values are entered directly by the user are automatically associated with any decision table which directly or indirectly uses them and, unless it is specified otherwise, a value will be requested for the variable each time the table is used. If the value of a relevant variable is not being queried, either: use the scratch case study (case 1) and execute the table, or use the query-don't query option on the RUN MENU and then execute the table.

Is the value of the variable to be queried each run ? (Y/N) - N ↵

:: enter a set value for this variable
(up to 12 digits, including a compulsory decimal point and optional 2 digit exponent)
2.0 ↵
As a result of this change the values of the following variables have been set to void -
Y

Note that you may vary the status of this variable from one case study to another

RUN MENU

Do you wish to -

1 select and run a program
2 re-run the last program
3 select a case study
4 save current case study
5 change variable status to query/don't query each run
6 preset (override) the value of a variable normally
calculated by the program. (Resets after next run)
Select your case study BEFORE using override.
7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry
0 return to main menu

:: enter number of choice - 5 ↵

:: enter name of variable - ONE ↵

INFORMATION :- variables whose values are entered directly by the user are automatically associated with any decision table which directly or indirectly uses them and, unless it is specified otherwise, a value will be requested for the variable each time the table is used. If the value of a relevant variable is not being queried, either: use the scratch case study (case 1) and execute the table, or use the query-don't query option on the RUN MENU and then execute the table.

Is the value of the variable to be queried each run ? (Y/N) - N ↵

:: enter a set value for this variable
(up to 12 digits, including a compulsory decimal point and optional 2 digit exponent)
1.0 ↵
As a result of this change the values of the
following variables have been set to void -
I

Note that you may vary the status of this variable
from one case study to another

RUN MENU

Do you wish to -

1 select and run a program
2 re-run the last program
3 select a case study
4 save current case study
5 change variable status to query/don't query each run
6 preset (override) the value of a variable normally
calculated by the program. (Resets after next run)
Select your case study BEFORE using override.
7 view a decision table
or the current value of a variable
or all equations associated with a variable
or a look-up table entry
0 return to main menu

:: enter number of choice - 5 ↵

:: enter name of variable - FOUR ↵

INFORMATION :- variables whose values are entered directly by the user are automatically associated with any decision table which directly or indirectly uses them and, unless it is specified otherwise, a value will be requested for the variable each time the table is used. If the value of a relevant variable is not being queried, either: use the scratch case study (case 1) and execute the table, or use the query-don't query option on the RUN MENU and then execute the table.

Is the value of the variable to be queried each run ? (Y/N) - N ↵

:: enter a set value for this variable
(up to 12 digits, including a compulsory decimal point and optional 2 digit exponent)
4.0 ↵

Any data which would've been affected by this change has already been set to void

Note that you may vary the status of this variable from one case study to another

```
RUN MENU
--------

Do you wish to -

1  select and run a program
2  re-run the last program
3  select a case study
4  save current case study
5  change variable status to query/don't query each run
6  preset (override) the value of a variable normally
   calculated by the program. (Resets after next run)
   Select your case study BEFORE using override.
7  view a decision table
        or the current value of a variable
        or all equations associated with a variable
        or a look-up table entry
0  return to main menu

:: enter number of choice -   / ↵


The following is a list of all decision tables :-

table  1     BEAM BENDING
table  2     BENDING MOMENT


:: enter number of decision table -  / ↵


The current value of SECTION NUMBER       is     100.E-02
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> 2.0 ↵


Any data which would've been affected by this change
has already been set to void
```

```
The current value of SY                    is     240.E+06 MPA
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN>  350.0E+06 ↵


Any data which would've been affected by this change
has already been set to void



The current value of LEFT END FREE         is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END FIXED        is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END GUIDED       is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of LEFT END SS           is Y
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵


The current value of RIGHT END FIXED       is N
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN> ↵
```

```
The current value of RIGHT END SS          is Y
If you wish to change this value enter the new value
(either Y or N) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN>  ↵


The current value of MASS                  is     500.E+01 KG
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN>  ↵


The current value of L                     is     600.E-02 M
If you wish to change this value enter the new value
(up to 12 digits including a compulsory decimal point
and optional 2 digit exponent) and press <RETURN>
If no change required press <RETURN> with no entry

:: enter new value or press <RETURN>  ↵


[table BENDING MOMENT                 , table no.  2, rule no.  5 applies]


[table BEAM BENDING                   , table no.  1, rule no.  1 applies]
BENDING STRESS OK


[table BEAM BENDING                   , table no.  1, rule no.  1 applies]
SB                  =     311.E+06 MPA


[table BEAM BENDING                   , table no.  1, rule no.  1 applies]


[table BEAM BENDING                   , table no.  1, rule no.  1 applies]
MS                  =     143.E-03


:: do you want the values of designer controlled  variables
written to the output file?  (Y/N) - Y ↵

Do you wish to DELETE the output from this LATEST
AND any PREVIOUS runs  (Y/N) - N ↵
```

```
RUN MENU
--------

Do you wish to -

1  select and run a program
2  re-run the last program
3  select a case study
4  save current case study
5  change variable status to query/don't query each run
6  preset (override) the value of a variable normally
   calculated by the program. (Resets after next run)
   Select your case study BEFORE using override.
7  view a decision table
        or the current value of a variable
        or all equations associated with a variable
        or a look-up table entry
0  return to main menu

:: enter number of choice - 4 ↵
case 2     Vacant                    .
case 3     Vacant                    .
case 4     Vacant                    .
case 5     Vacant                    .
case 6     Vacant                    .


:: enter case number data is to be filed under ' (0 to abort) - 4 ↵
:: enter a title for case study 4 -
BEAM MK II ↵

RUN MENU
--------

Do you wish to -

1  select and run a program
2  re-run the last program
3  select a case study
4  save current case study
5  change variable status to query/don't query each run
6  preset (override) the value of a variable normally
   calculated by the program. (Resets after next run)
   Select your case study BEFORE using override.
7  view a decision table
        or the current value of a variable
        or all equations associated with a variable
        or a look-up table entry
0  return to main menu

:: enter number of choice - 0 ↵
```

PROGRAM RUN 2

```
REMINDER - save your current case study (option 4)
           if it is required, before returning.

:: do you wish to return to the main menu?  (Y/N)- Y ↵


                    MAIN MENU

-------------------------------------------------

Do you wish to :-


1  Run an existing program or view data, tables etc
2  Modify an existing program or data
3  Create a new program
4  Quit

:: enter number of choice and press <RETURN>  4 ↵


To obtain your printout of output generated this
session type PRINT OUTPUT.DAT when prompt appears.

The output file will be DELETED on commencing the next run
```

```
[table BENDING MOMENT                , table no.  2, rule no.  5 applies]


[table BEAM BENDING                  , table no.  1, rule no.  1 applies]
BENDING STRESS OK

[table BEAM BENDING                  , table no.  1, rule no.  1 applies]
SB                  =     311.E+06 MPA

[table BEAM BENDING                  , table no.  1, rule no.  1 applies]


[table BEAM BENDING                  , table no.  1, rule no.  1 applies]
MS                  =     143.E-03

The following are the values of the variables set for this run :-
SECTION NUMBER      =      200.E-02
SY                  =      355.E+06 MPA
LEFT END FREE       =      000.E+00
LEFT END FIXED      =      000.E+00
LEFT END GUIDED     =      000.E+00
LEFT END SS         =      100.E-02
RIGHT END FIXED     =      000.E+00
RIGHT END SS        =      100.E-02
MASS                =      500.E+01 KG
L                   =      600.E-02 M
```